Sistemas Embarcados

Sistemas embarcados

Cristiano Marçal Toniolo

Sumário

Palavras do autor

Olá, aluno!

Seja bem-vindo à disciplina de Sistemas Embarcados para os cursos que envolvem tecnologia e automação de sistemas. Nesta disciplina, aprenderemos por que os sistemas embarcados são tão importantes para o desenvolvimento de projeto de automação, pois nos dias de hoje, podemos encontrar muitos produtos que têm um sistema embarcado dentro deles, o qual pode acessar a internet e buscar informações e parâmetros para resolver um determinado problema.

Esta disciplina mostrará todos os conceitos que deverão ser entendidos por você, aluno, para saber como um sistema embarcado funciona e quais são suas características principais. Com isso, nosso objetivo será o de levar você a aprender os conceitos fundamentais para o desenvolvimento de um projeto envolvendo sistemas embarcados. Como consequência disso, trabalharemos as competências referentes a entender o que são os sistemas embarcados, suas características e seus componentes, e com os conceitos iniciais apresentados, saber quais são as habilidades necessárias para o desenvolvimento de projetos para sistemas embarcados e as tecnologias que o mercado oferece para que um projeto deste tipo possa ser desenvolvido.

Assim, teremos como base as nossas unidades de ensino, que nos permitem delinear de forma mais específica o que será mostrado na disciplina.

Na Unidade 1, você terá contato com os conceitos de introdução a sistemas embarcados, com as características dos sistemas embarcados e, para finalizar, serão apresentados os hardwares para sistemas embarcados.

Na Unidade 2, você conhecerá sobre dispositivos, que serão introduzidos através dos conceitos de sistemas operacionais embarcados; depois, verá os sistemas de arquivos embarcados e, para finalizar, os dispositivos de armazenamento para sistemas embarcados que podem ser encontrados no mercado.

Na Unidade 3, veremos as configurações relacionadas aos sistemas embarcados que abrangem a configuração do sistema de arquivos, o carregador de inicialização de sistemas

embarcados e, finalmente, como é feita uma configuração de redes para sistemas embarcados.

Na Unidade 4, veremos como os sistemas de tempo real, seus sensores e atuadores trabalham e, para isso, aprenderemos sobre as ferramentas de depuração de sistemas embarcados e quais são os dispositivos embarcados para tempo real, e para que tudo isso fique bem esclarecido, aprenderemos também, quais sensores e atuadores embarcados podem ser usados nos projetos que teremos pela frente.

Dessa forma, queremos que você, aluno, tenha um excelente aprendizado e que os conceitos e os materiais aqui apresentados ajudem no seu crescimento profissional dentro e fora da Faculdade. Nós, professores, estaremos disponíveis para ajudá-lo a ter o melhor ensinamento possível, Para isso, você deve ter curiosidade e comprometimento!

Fundamentos gerais sobre sistemas embarcados

Convite ao estudo

Olá, aluno! Nesta unidade de ensino, trataremos do assunto “Introdução aos sistemas embarcados”, para que você tenha base conceitual do que são os sistemas embarcados. Esses tipos de sistemas têm aplicação onde os requisitos são muito específicos, tendo a função de realizar atividades que estão predefinidas e embarcadas em componentes cada vez menores e com menor consumo de energia, tornando o produto final cada vez menor, em que se pode acessar a internet (nos dias de hoje) e ter um custo relativamente barato em relação aos computadores de uso pessoal. Atualmente, podemos encontrar estes tipos de sistemas na porta de entrada de uma agência bancária (portas giratórias), em marcapassos, nos radares das rodovias, entre outros.

Esses equipamentos possuem um sistema embarcado dentro deles, ou seja, um sistema embarcado é um sistema de computador que vem inserido em um equipamento sem que um usuário possa alterar suas configurações de forma direta. Dessa forma, como visto no início do nosso material, nesta seção, conheceremos os fundamentos de um sistema embarcado e desenvolveremos as habilidades necessárias para que você possa desenvolver um projeto com suas tecnologias mais recentes, que podem se referir a sistemas embarcados. Para que possamos entender bem os conceitos que serão apresentados, trabalharemos o contexto de aprendizagem de uma empresa do ramo industrial, que está há vários anos no mercado. Ela viu seus concorrentes saírem na frente no conceito de inovação, e todos já estão com seus processos de produção automatizados.

Assim, para que ela volte a concorrer em igualdade de condições com seus concorrentes, necessita elaborar e implantar um projeto que contemple o uso de sistemas embarcados e automatização da linha de produção. Essa automação se dará nos equipamentos mecânicos que demandam mais força de trabalho, assim, deixá-los automatizados poderá garantir maior rendimento e qualidade na produção da empresa. No entanto, surge um grande problema: a empresa não tem o know-how suficiente para executar o projeto. Para isso, você foi contratado para executar esse projeto desde seu início até sua implantação e, também, treinar toda a equipe da empresa para os conceitos iniciais sobre sistemas embarcados. Você deverá treinar a equipe de produção da empresa para que, após a implantação do projeto, ela possa dar continuidade à manutenção dos sistemas por conta própria. Para entender qual o nível de conhecimento das pessoas envolvidas no projeto, temos que perguntar: o que eles entendem sobre sistemas embarcados? O que esperam de resultado para uma empresa automatizada? Quais são as vantagens que terão ao mudar o processo da empresa de manual para automatizado?

Veja como é interessante o assunto sistemas embarcados, então, para que você se situe no que será visto nesta unidade, abordaremos os fundamentos relacionados aos sistemas embarcados, pois eles diferem bastante em relação a sistemas de computação atuais; depois, abordaremos informações referentes aos sistemas embarcados:

- Sistemas embarcados de tempo real em termos conceituais, pois no final do curso, a abordagem será aplicar esses sistemas de forma prática.

- Saberemos qual é a arquitetura de software de um sistema embarcado, para que possamos entender quais são as diferenças em relação aos sistemas de programação comuns.

- Aplicações dos sistemas embarcados nos dias de hoje.

Seção 1.1

Introdução a sistemas embarcados

Diálogo aberto

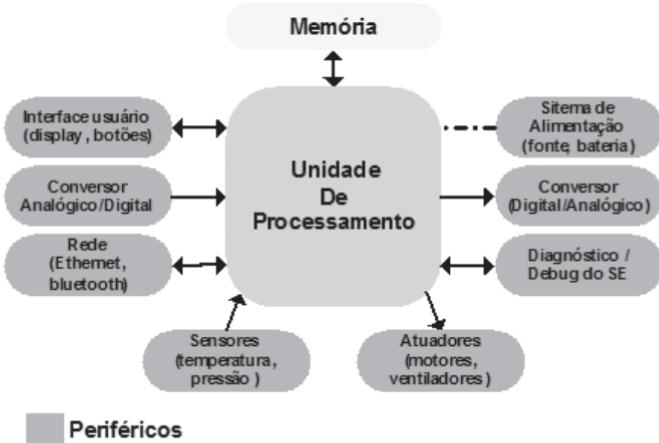
Você foi selecionado para desenvolver um projeto de automação com sistemas embarcados para uma empresa do ramo industrial e tem, como ponto de partida, que mostrar para os envolvidos neste projeto quais são os conceitos iniciais relacionados a sistemas embarcados, tais como os fundamentos de sistemas embarcados, o que são sistemas embarcados em tempo real, quais são os tipos de arquitetura de software em sistemas embarcados e como aplicar os projetos de sistemas embarcados e seu uso em dispositivos móveis.

Aplicação de sistemas embarcados e seu uso em dispositivos móveis

Você deve fazer um projeto inicial que contemplará os seguintes itens: quais são os fundamentos relacionados a sistemas embarcados? Qual é a diferença entre sistemas embarcados e sistemas embarcados de tempo real? Quais são as arquiteturas de softwares embarcados e suas aplicações?

Para que esta proposta seja entregue ao nosso cliente, você deverá elaborar um relatório, em que constará todos os tópicos estudados nesta unidade e, assim, explicar para os envolvidos no projeto quais são os fundamentos necessários para que possamos avançar no projeto de um sistema embarcado, ou seja, no final do relatório, deverá constar uma conclusão de sua parte sobre o que é mais relevante para as pessoas envolvidas no projeto, a fim de evitar que informações diferentes possam aparecer nos próximos passos do desenvolvimento.

Este relatório terá que respeitar as normas ABNT e deverão constar capa, sumário, desenvolvimento do tema, conclusão e referências bibliográficas.

Não pode faltar

Sistemas embarcados podem ser encontrados em diversos equipamentos e em diversas áreas de atuação. Para isso, nesta unidade, abordaremos os conceitos referentes a sistemas embarcados, suas funcionalidades e aplicações. Antes, definiremos o que são sistemas embarcados e seus conceitos principais. Segundo Oliveira (2017, p. 40), são sistemas que "começaram a se desenvolver e ocupar tarefas que antes eram feitas de forma manual. Alguns desses componentes começaram a se destacar na automatização dos dispositivos e sistemas por agregar mais funcionalidade no próprio componente", ou seja, qualquer dispositivo que tenha um sistema que possa ser controlado através de entrada de dados automática ou por meio de interface, como teclas, teclados e controle remoto, são exemplos de sistemas embarcados. Eles estão divididos em hardwares, que têm como objetivo executar comandos através de atuadores, que transformam os comandos enviados a eles em movimentos, funcionamento de motores, entre outros mecanismos que têm interface com um programa de computador (software), o qual recebe as entradas de dados através de sensores e estes dados são transformados em sinais digitais que alimentam os sistemas para tomar decisões, e através destas decisões, enviam os comandos para os atuadores corretos, a fim de executarem a ação para que foram programados.

Você pode estar pensando: qual é a diferença entre sistemas embarcados e sistemas de uso geral? Suponha que você tem um computador pessoal ou um notebook. Esses equipamentos trabalham com quantos dados ao mesmo tempo? Quais são os objetivos desses equipamentos?

Respondendo a essas questões, os equipamentos citados trabalham com grande quantidade de dados e são multiobjetivos, ou seja, podem ser usados para resolver uma quantidade enorme de situações onde seu uso é imprescindível. Têm grande poder de processamento, tanto que usam um ou vários microprocessadores para que determinados objetivos possam ser alcançados, ou seja, podemos perceber que os sistemas de uso geral necessitam de maior capacidade de processamento. Já no caso dos sistemas embarcados, o conceito é o oposto, possuem baixa capacidade de processamento e sua aplicação pode ser para uma quantidade pequena de objetivos ou até para resolver um único objetivo.

Para que possamos ilustrar o que vem a ser um projeto de sistemas embarcados, a Figura 1.1 mostra todos os componentes envolvidos no projeto e que devem ser respeitados no momento da elaboração do projeto.

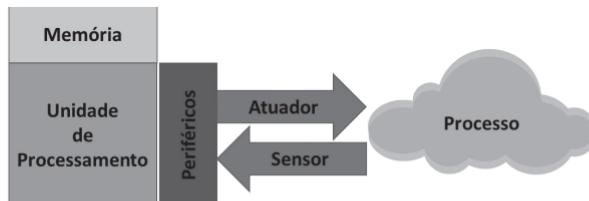



Figura 1.1 | Modelo de sistema embarcado com suas subdivisões

Fonte: adaptada de Delai (2013).

Podemos observar, na Figura 1.1, que o projeto de um sistema embarcado está dividido em três grandes partes, que são: unidade de processamento, na qual os dados serão processados de acordo com as entradas feitas através dos sensores. Essa unidade executa cálculos, faz o tratamento de eventos e também pode ser programada para tomar decisões. A segunda é a memória, em que os dados e as instruções são trabalhados através de operações e também pode armazenar dados. E, por fim, temos os periféricos, que fazem a interface da unidade de processamento com o ambiente externo, trazendo e enviando informações para essa unidade. A seguir, mostraremos uma figura que exibe como os periféricos fazem a interface com a unidade de processamento.

Figura 1.2 | Periféricos, atuadores e sensores

Fonte: elaborada pelo autor.

Na Figura 1.2, temos os equipamentos que fazem a interface entre os processos, que são as funções que cada equipamento executa durante um determinado tempo, e os periféricos. Antes de enviar os dados para a unidade de processamento de um sistema embarcado, os sensores, que são componentes eletrônicos que captam informações do ambiente, transformam em sinais elétricos e enviam para o sistema. Os atuadores são componentes que executam uma determinada resposta resultante de um processamento realizado pelo sistema embarcado. Podemos observar que os sensores estão apontados para o processo, ou seja, têm a função de fazer com que os comandos sejam atuados através deles para que o processo possa ocorrer de forma automática. Já os sensores estão apontados na direção dos periféricos, que têm o objetivo de receber esses dados e enviá-los para a unidade de processamento.

Assim, podemos perguntar: esses sistemas embarcados têm que tipo de resposta? Dependem de algum tempo para receber os dados e processá-los ou podem fazer isso de forma instantânea? Como se dá o funcionamento desses sistemas em caso de necessidade urgente, ou seja, precisando de uma resposta para que atuem de forma instantânea?

Nesse caso, temos os sistemas embarcados de tempo real, ou seja, sistemas que têm a capacidade de receber dados e processá-los de forma instantânea, sem a necessidade de um tempo para que a unidade de processamento e a memória executem os processamentos e depois enviem as respostas para os atuadores.

Sistemas embarcados de tempo real são sistemas "que gerenciam os recursos de um sistema computacional, com o objetivo de garantir que todos os eventos sejam atendidos dentro de suas restrições de tempo, e gerenciados da forma mais eficiente possível" (PRADO, 2010, [s.p.])

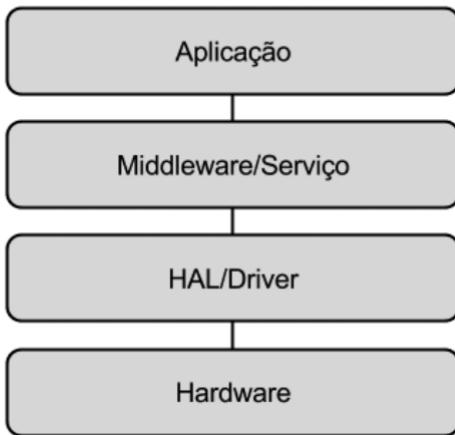
Podemos notar que esse tipo de sistema difere dos sistemas embarcados por causa da variável tempo, ou seja, para que um determinado problema seja resolvido, está relacionado ao tempo que deve ser parametrizado no sistema e que responda às entradas de dados com relação às suas restrições, fazendo com que as respostas sejam executadas com a melhor performance possível.

Para ilustrar o que os sistemas embarcados de tempo real podem fazer, mostraremos alguns exemplos de aplicações que eles podem ajudar, como um sistema de controle de temperatura, no qual os sensores inserem os dados sobre a temperatura, que se estiver em nível crítico, fará com que o microcontrolador responda de forma imediata, fazendo

com que os atuadores façam a temperatura chegar a níveis aceitáveis. Esse sistema pode ser chamado de sistema embarcado de tempo real altamente restritivo devido à sua atuação. Outro exemplo que podemos citar está relacionado a um sistema de radar aeroespacial, o qual recebe dados referentes ao posicionamento de aeronaves, em que, se o sistema não levar estas informações de forma relevante, poderá causar acidentes aéreos de grande escala. Monitores cardíacos de UTIs devem levar em consideração a variação dos batimentos cardíacos dos pacientes, se houver uma variação muito grande entre um batimento e outro, devem ativar o alarme em poucos segundos, para que os médicos possam executar os procedimentos padrão da UTI.

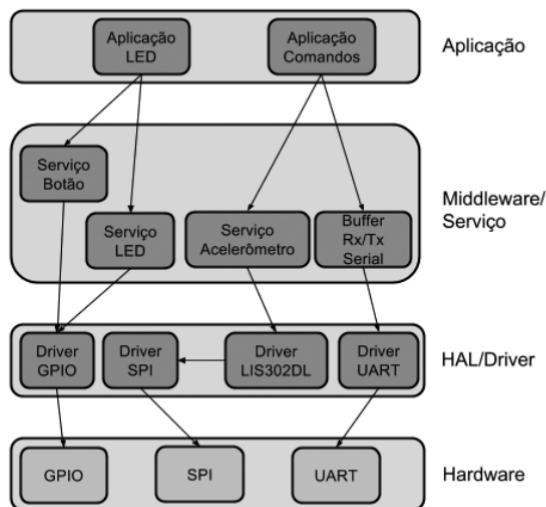
Figura 1.3 | Monitor de batimentos cardíacos

Fonte: Puhlmann (2014, [s.p.]).

Assimile

Sistemas embarcados são sistemas que vêm integrados dentro de um equipamento ou dispositivo. Sistema embarcado de tempo real permite que um sistema embarcado possa reagir a uma determinada situação em um tempo preestabelecido, de acordo com a aplicação ou o objetivo, como um sistema embarcado de tempo real para monitorar os batimentos cardíacos de um paciente, conforme pode ser visto na Figura 1.3.

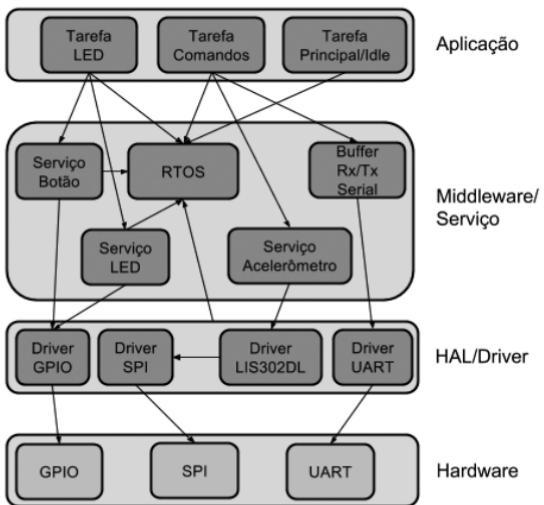
E como deve ser a arquitetura de software para sistemas embarcados? Quais, suas características? Os softwares para sistemas embarcados estão relacionados a uma estrutura que permite que os elementos do software se relacionem com os elementos externos do sistema embarcado, que são os hardwares, ou seja, são a interação entre hardware e software para

que os projetos de sistemas embarcados funcionem. Dessa forma, vale ressaltar que, nos dias de hoje, pode-se usar *frameworks*, que permitem, através de seus conjuntos de componentes computacionais, fazer a relação entre os componentes utilizados nos sistemas embarcados. Assim, podemos levar essa interação para o modo visual, no qual a arquitetura de software é um grafo que tem como arestas os componentes, e os vértices são as ligações entre esses componentes. Para isso, mostraremos como é a definição de arquitetura de software para sistemas embarcados mais utilizada dentre os diversos padrões de arquitetura existentes, que é a de sistema em camadas.

Figura 1.4 | Sistema em camadas

Fonte: adaptada de Rossi (2015, [s.p.]).

Podemos notar, na Figura 1.4, que as camadas que compõem o sistema embarcado são as de Aplicação, que estão relacionadas a como o usuário final interagirá com nosso sistema, ou seja, quais são os componentes visuais que poderão ser mostrados para este usuário. Depois, vem a camada de Middleware/Serviço, que permite que entradas feitas pelos usuários na camada de aplicação possam ser executadas pelos componentes, tais como leds, botões, entre outros. A próxima camada é a de Driver, responsável por tratar dos hardwares relacionados aos componentes que estão sendo utilizados no nosso sistema. E, para finalizar, temos a camada de Hardware, que são as placas físicas que foram implementadas no sistema embarcado. Esse padrão de sistema em camadas tem como objetivo ajudar a fazer com que os projetos

embarcados sejam construídos com qualidade, pois permite manter uma certa modularidade, coesão e encapsulamento, permitindo organização de projeto. Exemplo: uma parte da aplicação não pode acessar um driver sem antes passar pela camada de Middleware, que é responsável por resolver problemas de driver. Um exemplo de forma prática de como este tipo de estrutura é utilizado pode ser visto na figura a seguir.

Figura 1.5 | Arquitetura de um sistema embarcado sem RTOS

Fonte: adaptada de Rossi (2015, [s.p.]).

Na Figura 1.5, podemos notar as quatro camadas mencionadas anteriormente e o que cada uma delas está responsável por controlar. Na camada de Aplicação, estão as aplicações de Led e Comandos, ou seja, neste caso, podemos ter duas formas de interface com o usuário, em que os comandos são dados pelo usuário, e os Leds podem sinalizar saída de dados ou, também, se algum comando acionado deveria mostrar que estava funcionando através de um Led. Na camada de Middleware/Serviço, podemos notar os serviços relacionados aos Leds por meio de botões e os serviços relacionados aos comandos por meio de um acelerômetro e entrada serial.

Na camada de Driver, podemos ver os drivers dos dispositivos que estão sendo usados no projeto, como GPIO (*General Purpose Input/Output*), que está relacionado à entrada de saída de dados do sistema embarcado. E, por fim, a camada de Hardware nos permite verificar quais são os hardwares usados no sistema embarcado e como eles estão ligados

com as camadas anteriores. Isso sem que seja um sistema embarcado em tempo real. Para este tipo de sistema, veremos a Figura 1.6, que mostra suas camadas.

Figura 1.6 | Arquitetura de sistema embarcado com RTOS

Fonte: adaptada de Rossi (2015, [s.p.])

As diferenças entre o diagrama de sistema em camadas sem RTOS e com RTOS está no fato de que na camada de aplicação aparece a Tarefa Principal, ou seja, esse tipo de sistema está preocupado em resolver as tarefas que têm maior prioridade naquele momento, a fim de otimizar o tempo de resposta para atuação. Na camada de Middleware, temos o bloco de RTOS (*Real Time Operating System*), que está relacionado aos inputs do sistema de tempo real. As camadas de Driver e Hardware continuam da mesma forma, pois a camada anterior tem a responsabilidade de ter as funções de programação referentes ao tratamento dos eventos que chegarão ao sistema embarcado de tempo real.

Reflita

Por que os sistemas embarcados de tempo real merecem mais atenção no desenvolvimento de um projeto do que os sistemas embarcados que podemos chamar de "comuns"? Ou essa atenção deverá ser tomada para ambos os tipos de sistemas embarcados?

Até aqui tivemos uma visão dos conceitos sobre sistemas embarcados relacionados aos seus fundamentos, assim como dos sistemas embarcados em tempo real e quais são os tipos de arquiteturas de software em sistemas embarcados. Dessa forma, agora veremos, exemplos de como e quais são as aplicações dos sistemas embarcados que poderemos encontrar no nosso dia a dia.

Pesquise mais

Sistemas embarcados na robótica. Disponível em: <https://www.youtube.com/watch?v=FvX33N1JOXY>. Acesso em: 9 out. 2017. Nesse vídeo, são mostrados exemplos de aplicação de sistemas embarcados na robótica atual e alguns robôs em funcionamento.

Desenvolvedor de software embarcado. Disponível em: <https://www.youtube.com/watch?v=fiTBVb1hBXE>. Acesso em: 9 out. 2017. Nesse vídeo, são passadas algumas dicas para que se possa começar a se aventurar no desenvolvimento de software para sistemas embarcados.

Começaremos citando sistemas embarcados utilizados nos automóveis. Atualmente temos várias montadoras desenvolvendo sistemas embarcados e implantando nos seus carros mais vendidos. Podemos citar as seguintes montadoras: Volkswagen, Audi, Toyota, BMW, Mercedes, entre outras. Quais modelos possuem sistemas embarcados? Alguns modelos são Golf, Audi A3, Corola, BMW Série 4, Mercedes Cabriolet, entre outros, pois são inúmeros modelos por marca. E os sistemas embarcados estão controlando o GPS do carro, o sistema de freios ABS, o Air Bag, o controle de tração, o controle do motor e da injeção eletrônica, os alarmes, entre outros sistemas que um automóvel pode ter. Mais aplicações de sistemas embarcados estão relacionados à área da saúde, em que podemos encontrar os monitores cardíacos das UTIs, já citados anteriormente nesta unidade, e os marcapassos, que atualmente têm um sistema de telemetria que permite comunicação sem fio entre o dispositivo e o programador ou monitor residencial, para que o paciente tenha um excelente acompanhamento.

Exemplificando

Outros exemplos de aplicação de sistemas embarcados são os smartphones, as TVs inteligentes, os relógios, os roteadores e a robótica. Todas as aplicações permitem a evolução da tecnologia, para que seja

utilizada em benefícios para a humanidade. Assim, devemos ter em mente como e quando utilizar nosso conhecimento para trazer retorno de sucesso às pessoas.

Bons estudos e até a próxima unidade!

Sem medo de errar

Nosso "Sem medo de errar" nos remete à situação-problema do início da unidade, em que a empresa de produção selecionou você para desenvolver um projeto de automação com sistemas embarcados para a empresa do ramo industrial, tendo, como ponto de partida, que mostrar para os envolvidos neste projeto quais são os conceitos iniciais relacionados a sistemas embarcados. Assim, você deve fazer um projeto inicial, o qual contemplará os seguintes itens: quais são os fundamentos relacionados a sistemas embarcados, a diferença entre sistemas embarcados e sistemas embarcados de tempo real, as arquiteturas de softwares embarcados e suas aplicações. Você começou o seu trabalho e teve como resultado os seguintes conceitos que serão repassados para o pessoal diretamente envolvido no projeto em forma de relatório:

a. Fundamentos sobre sistemas embarcados.

b. Quais são as diferenças entre sistemas embarcados e sistemas embarcados de tempo real?

c. Quais são as arquiteturas de softwares embarcados e aplicações?

Assim, pode mostrar para a empresa que os conceitos iniciais sobre sistemas embarcados devem contemplar todos os requisitos necessários para que os próximos passos do projeto sejam feitos de forma sucinta e clara, pois qualquer dúvida sobre o que são os sistemas embarcados poderá impactar na escolha e no desenvolvimento do sistema.

A situação-problema (SP) da seção:

Portanto, deverá entregar um relatório intitulado "Conceitos iniciais sobre sistemas embarcados".

Deverá constar na capa quem foram os integrantes do projeto que criaram o relatório.

Deverá constar um índice/sumário que permita uma rápida busca por tópico, se necessário.

O desenvolvimento dos conceitos pesquisados.

Uma conclusão sobre o que a empresa poderá pensar sobre sistemas embarcados em seu ramo de atuação e, claro, as referências de onde as informações foram encontradas.

Avançando na prática

Por que automatizar um processo que já funciona corretamente?

Descrição da situação-problema

Um novo cliente entrou em contato para automatizar uma linha de produção de vidros de remédios para a indústria farmacêutica. Nesta linha de produção, o processo de extrusão dos frascos de vidro é feito manualmente pelos funcionários da empresa. Assim, você deverá levantar os dados referentes a essa linha de produção, seus equipamentos e sugerir a implantação de um sistema embarcado para os equipamentos que necessitem serem automatizados. Esse levantamento deverá ser entregue em formato de relatório, em que serão avaliados os itens sobre os conceitos de sistemas embarcados, como foi o levantamento dos pontos e equipamentos a serem automatizados e quais os critérios para escolha de um sistema embarcado.

Resolução da situação-problema

Relatório sobre automatização de linha de produção de frascos de remédio

Aqui, devemos levar em conta uma linha de produção em que todos os processos são mecânicos, ou seja, tem sempre a interferência de um funcionário para que possa ocorrer de forma correta e atingir os níveis de qualidade e metas da empresa. Dessa forma, faremos o relatório para uma linha de produção de frascos de remédio, em que temos o controle de recebimento de matéria-prima, que é manual. Após o recebimento da matéria-prima, deve-se armazenar no estoque da empresa para que seja colocada nas

prateleiras corretas para uso posterior. A matéria-prima, atualmente, é colocada nas máquinas de forma manual. Depois de processada a primeira vez, ela continua para a próxima máquina, ainda de forma manual, ou seja, nada de automação. E como a empresa tem cinco passos nos processos de fabricação de seu produto, até sua embalagem, todos os processos continuam sendo manuais.

Dessa forma, foi levantado que:

1- O processo de envio da matéria-prima para sua primeira fase de processamento deverá ser automatizado.

2- Após a primeira fase, ainda não conseguiremos automatizar o envio para a segunda fase.

3- A segunda fase de processamento será automatizada.

4- Após a segunda fase, ainda não conseguiremos automatizar o envio para a terceira fase.

5- A terceira fase de processamento será automatizada.

6- Após a terceira fase, ainda não conseguiremos automatizar o envio para a quarta fase.

7- A quarta fase de processamento será automatizada.

8- Após a quarta fase, ainda não conseguiremos automatizar o envio para a quinta fase.

9- A quinta fase de processamento será automatizada, e nesta fase é feita a embalagem do produto.

10- Nesta fase de embalagem, os produtos podem ser embalados automaticamente, sem a necessidade de pessoal para isso.

Os critérios adotados foram os seguintes: nos processos feitos com um equipamento mecânico, pode-se fazer a automatização, já que é permitido o uso de sensores e atuadores para que eles possam ser automatizados. Já os processos de transferência de uma máquina para outra, não vemos a necessidade ainda de se automatizar, pela necessidade de se ter uma pessoa responsável por controlar cada máquina e seus processos.

Faça valer a pena

1. Nos dias atuais, os sistemas embarcados estão presentes em diversos equipamentos. Com isso, uma grande oportunidade para os profissionais de

computação surgiu, pois muitas pessoas que trabalham nesta área não têm qualificação suficiente para desenvolver tal função.

Assim, quais são as habilidades necessárias que um profissional qualificado deve ter para que possa assumir uma responsabilidade em projetos de sistemas embarcados nos dias de hoje?

a) Saber de sistemas embarcados e ter as habilidades que qualquer desenvolvedor de sistemas tem para que o projeto possa ser embarcado em um equipamento.

b) Ter a noção de que desenvolver para sistemas embarcados é muito diferente de desenvolvimento de sistemas para computação.

c) Saber os fundamentos de sistemas embarcados, suas habilidades e desenvolver projetos para sistemas embarcados.

d) Conhecer e compreender os fundamentos de sistemas embarcados, com habilidades para desenvolvimento de projetos e tecnologias para estes sistemas.

e) Conhecer e saber como funcionam os sistemas embarcados, com habilidades para criar projetos de sistemas e não usar novas tecnologias.

2. Sistemas embarcados diferem dos sistemas embarcados de tempo real por uma questão que permite aos sistemas de tempo real terem uma resposta mais rápida para um determinado input vindo dos sensores, ou seja, este tipo de sistema tende a ser mais eficiente que o primeiro.

Com base no que foi apresentado, qual é a questão que permite aos sistemas embarcados de tempo real serem mais eficientes em alguns projetos do que os sistemas embarcados?

a) Está relacionado à entrada de dados, pois este parâmetro permite que ele retorne uma resposta mais exata para o atuador.

b) Está relacionado à variável tempo, pois este parâmetro permite que ele retorne uma resposta mais exata para o atuador.

c) Está relacionado à variável tempo, pois este parâmetro permite que ele retorne uma resposta comum para o atuador.

d) Está relacionado à variável tempo, pois este parâmetro não tem relação com o retorno da resposta para o atuador.

e) Está relacionado à variável de entrada, pois este parâmetro permite que ele retorne uma resposta mais exata para o atuador.

3. Nos dias de hoje, temos inúmeros projetos que podem ser melhorados com o desenvolvimento de sistemas embarcados, entre eles, projetos relacionados à área da saúde, em que a tomada de decisão de um médico deve ser a mais exata possível para que se possa evitar problemas graves com seus pacientes, como em uma UTI.

Dessa forma, como um sistema embarcado poderá ajudar na tomada de decisão de um médico e sua equipe para salvar vidas dentro do hospital?

a) O sistema embarcado poderá mostrar os dados relacionados a batimentos cardíacos, para que a equipe médica execute os procedimentos de salvamento.

b) O sistema embarcado poderá enviar dados e mostrar os dados relacionados a batimentos cardíacos, para que a equipe médica execute os procedimentos comuns.

c) O sistema embarcado poderá enviar alarmes e mostrar os dados relacionados a batimentos cardíacos, para que a equipe médica execute os procedimentos de salvamento.

d) O sistema embarcado poderá enviar alarmes e não mostrar os dados relacionados a batimentos cardíacos, para que a equipe médica execute os procedimentos de salvamento.

e) O sistema embarcado apenas controla as informações e não faz mais nada além disso.

Seção 1.2

Características dos sistemas embarcados

Diálogo aberto

Olá, aluno! Seja bem-vindo à segunda seção da disciplina de Sistemas embarcados. Nesta seção, veremos sobre o que os sistemas embarcados podem fazer no nosso dia a dia e quais são as suas características, lembrando que, na Seção 1.1, você foi convidado a fazer parte do desenvolvimento de um projeto para uma empresa no ramo industrial, a qual necessita automatizar toda sua linha e processos de produção. Tudo isso pensando no mercado atual e nos seus concorrentes. Dessa forma, e já retomando a sua participação no projeto de automação, aqui, você deverá fazer um documento que mostre para as pessoas interessadas por parte da empresa quais são as principais características dos sistemas embarcados e suas categorias.

Assim, você realizará o levantamento das principais ferramentas disponíveis para a construção de sistemas embarcados. Para isso, devemos passar pelo aprendizado de como são os conceitos relacionados ao tema da unidade, que permitirá elencar todas as caraterísticas necessárias para que possamos elaborar um projeto de desenvolvimento e implantação de um sistema embarcado, tanto para a empresa desse cliente, quanto para qualquer outra empresa que possa vir a ser cliente.

Assim, com os dados levantados referentes aos conceitos apresentados nessa situação-problema e que serão esclarecidos na seção “Não pode faltar”, seu relatório poderá ser entregue aos integrantes da empresa contratante de seus serviços.

Não pode faltar

Caro aluno, nesta unidade, assimilaremos quais são os conceitos mais importantes referentes a sistemas embarcados. Veremos como as características serão influenciadas e/ou influenciarão no momento de uma escolha ou uma proposta de desenvolvimento livre de um sistema.

Principais características dos sistemas embarcados

Para aprofundar mais o seu conhecimento em sistemas embarcados, devemos lembrar dos conceitos apresentados na primeira seção, que abordou a diferença entre um sistema computacional não embarcado e um sistema embarcado, em que o primeiro tem como objetivo o uso em microcomputadores e notebooks, tendo como proposta o uso de usuários comuns para manipulação de aplicativos, como editores de texto, planilhas eletrônicas, entre outros. Já quando se fala em embarcados, devemos lembrar que o objetivo é baseado em funcionalidades específicas, que, na maioria das vezes, não tem a participação de usuários. Para isso, devemos lançar mão de algoritmos específicos e com aplicações complexas, além de elaborar *interfaces gráficas*, que possam dar resultados prontos a partir de dados recebidos por sensores e enviados para a *interface* e executados através de atuadores.

Outra grande característica está relacionada ao baixo custo de desenvolvimento de um projeto, pois por ter um sistema embarcado em um microcontrolador, os componentes que farão todo o trabalho podem ser diminuídos em relação a componentes usados em computadores, por exemplo. Também podemos citar como característica a forma que o sistema interage com o ambiente em que ele está inserido, pois existe a possibilidade de nenhuma interação dos usuários com o sistema, como as linhas de produção de refrigerantes nos dias de hoje, em que podemos estabelecer que, de todo o processo, uma grande parte da linha de produção é automatizada e sem a intervenção humana.

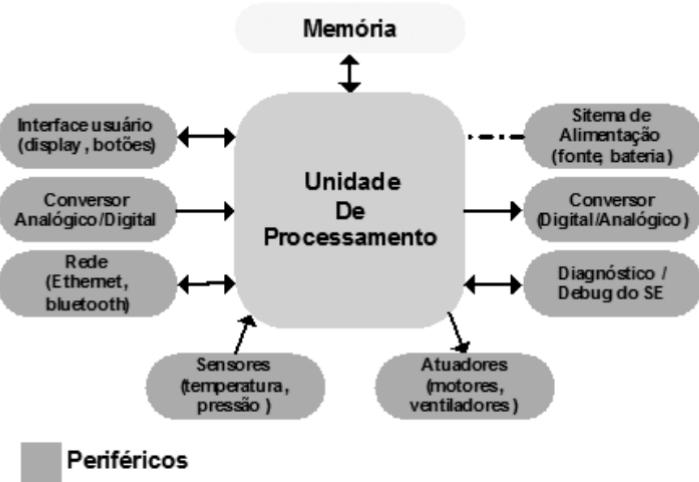
Para que isso seja possível, a integração entre sistema, sensores e atuadores deve ser bem planejada, podendo chegar a um grau de complexidade impressionante. Todas essas possibilidades remetem à possibilidade de usar esses sistemas em dispositivos que podem ser levados de um local para outro, o que podemos chamar de portabilidade, como os dispositivos de controle de processos, que são usados pelas empresas e permitem o acesso à internet ou à rede sem fio.

Esse tipo de equipamento é usado para obter dados de produção de várias máquinas em uma linha de produção. Dessa forma, podemos elencar outra característica, que é a análise das

propriedades não funcionais que outros tipos de sistemas não são possíveis de analisar, tais como o trabalho com sistemas de tempo real, sistemas de segurança, sistemas de tolerância a falhas, que são exemplos de sistemas em que qualquer erro ou dado inserido erroneamente causará um dano ou prejuízo de dimensões incalculáveis.

Como todas as aplicações que têm grande sucesso no mercado, os sistemas embarcados também têm algumas restrições que devem ser levadas em consideração no momento da análise e do projeto, como o tempo de duração de suas fontes de energia, pois dependendo de onde o sistema for aplicado, pode não existir alimentação de energia.

Outro ponto é com relação ao uso de memória para embarcar o sistema, pois como são inseridos em pequenas placas, a quantidade de memória é bastante limitada, fato que ajuda a explicar a característica referente à aplicação do sistema embarcado ser para objetivos bem específicos, e não de uso geral.

Assimile

Um sistema embarcado depende muito do sistema de alimentação pelo fato de ter tamanho reduzido, com pouca quantidade de memória e, na maioria das vezes, não pode deixar de funcionar porque outros sistemas dependerão dele, assim como em casos de uma rede de sensores que monitora uma área de mata para evitar incêndio. O grande impacto que aparece aqui é o da alimentação dos sensores, pois não há geração de energia e, assim, o uso de baterias garante o funcionamento por um determinado tempo.

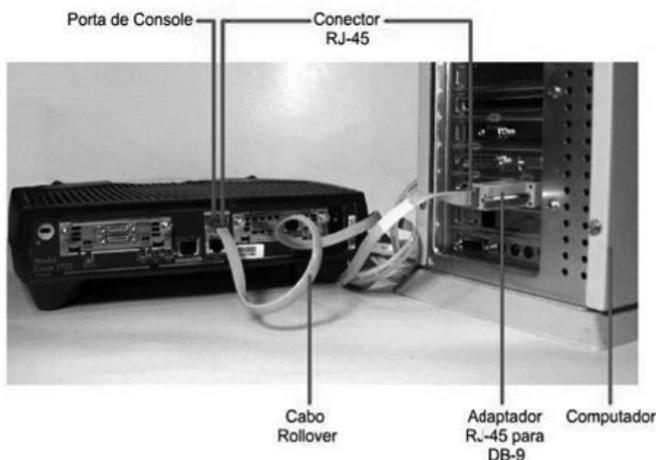
Categorizando os componentes de sistemas embarcados

Já que conceituamos os sistemas embarcados, agora, veremos quais são os componentes que fazem parte de uma estrutura genérica para esses sistemas, que por serem genéricos, podem ser encontrados na literatura ou em sites de empresas que desenvolvem projetos embarcados.

De acordo com Delai (2013), um projeto de sistema embarcado é composto pelos seguintes componentes, conforme mostra a Figura 1.7.

Figura 1.7 | Componentes de um sistema embarcado

Fonte: Delai ((2013, [s.p.])

Podemos notar que os componentes básicos para que um sistema embarcado possa ser projetado partem de uma unidade de processamento, que é o "coração" do projeto, pois todos os cálculos e processos são executados dentro da unidade de processamento. Depois, temos os componentes que interagem com essa unidade, a memória, que pode ser bem especificada e utilizada quando o projetista faz uso da documentação das memórias que vêm disponibilizadas para uso no mercado e onde são embarcados os sistemas que farão a interface com o ambiente externo. Após isso, temos um sistema de *interface* que pode interagir ou não com um usuário e pode ser composto por botões ou displays de toque, além de conversores, que fazem a tradução de dados analógicos e digitais para que o sistema receba como entrada ou envie como saída.

Reflita

Para sabermos como os sistemas embarcados funcionam, devemos aprender todos os conceitos básicos ou podemos ir diretamente aos exemplos práticos de projeto, como o uso de placas Arduino, sem sabermos o que são as portas lógicas e analógicas?

Um sistema embarcado, atualmente, pode interagir com a rede da empresa ou com a internet, ainda mais com a Internet das Coisas (*Internet of Things*), podendo acessar os dados diretamente da internet ou de outros equipamentos ligados na rede. Devemos usar os sensores e os atuadores que fazem a iteração com o ambiente externo, permitindo que os dados sejam inseridos no sistema (sensores), processados e, depois, possam retornar para o ambiente externo através dos atuadores, conforme vimos na Figura 1.7. O sistema de alimentação permite assegurar que o sistema possa estar disponível sempre que necessário, ou seja, esse sistema tem que ser bem pensado, pois caso o sistema embarcado seja de tempo real, ele não pode, em hipótese alguma, deixar de funcionar.

Em alguns sistemas embarcados, podemos ter um sistema de diagnóstico que, caso haja a necessidade ou tenha ocorrido um erro, um usuário possa inserir alguns comandos ou agir para regularizar essa situação. Esse é o caso do roteador Cisco da série 1700, que é dotado de um sistema embarcado que permite sua configuração e acesso através de comandos digitados em um *prompt* de comandos, permitindo que novas configurações sejam inseridas em caso de erro ou por algum tipo de atualização.

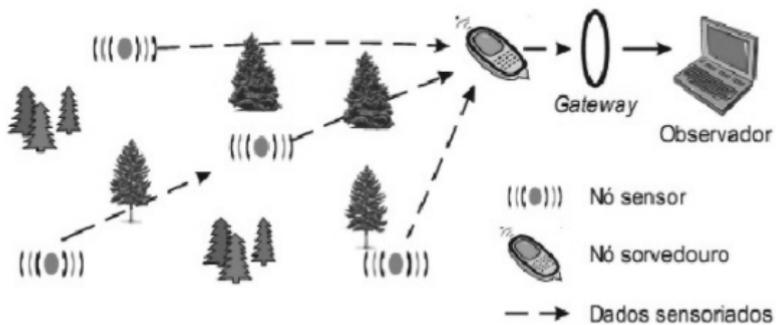
Figura 1.8 | Sistema embarcado do roteador da Cisco

Fonte: Cisco (2017, [s.p.]).

Figura 1.9 | Configuração de um sistema embarcado do roteador da Cisco

Fonte: Cisco (2017, [s.p.]).

Ferramentas para desenvolvimento de sistemas embarcados

Para o desenvolvimento de sistemas embarcados, devemos ter em mente quais ferramentas disponíveis no mercado podemos adotar e usar a contento. Assim, mostraremos algumas delas para que você, aluno, possa entender qual a gama de ferramentas que podemos usar e que nem tínhamos ideia de que existiam.

Assim como ocorre com os sistemas operacionais utilizados em computadores e notebooks, os sistemas embarcados possuem sistemas proprietários (pagos) e ferramentas livres (gratuitas). Um exemplo de sistema embarcado livre é o Symbian, que tem como característica ser aplicado em dispositivos móveis com suporte a *multithread*, ou seja, multitarefa. Ele pode ser usado para desenvolvimento de sistemas de tempo real, mas com uma restrição: funciona somente em processadores ARM; é usado em dispositivos das empresas Ericsson e Nokia e, atualmente, em mais empresas do ramo de tecnologia.

Outro sistema é o Sistema Operacional Windows CE, que permite o uso com várias versões de Windows para dispositivos móveis, como o Windows CE 6, que permite baixar o *kernel* da ferramenta para consulta do seu código-fonte gratuitamente. Pode ser programado usando uma ferramenta de desenvolvimento, que é o Visual Studio, com o pacote de desenvolvimento .Net Compact Framework, ou seja, ferramenta de desenvolvimento disponível no site da Microsoft de forma gratuita e que permite programação para este

tipo de plataforma. Windows Mobile é outra versão da ferramenta de desenvolvimento da Microsoft para que sejam desenvolvidos sistemas embarcados para smartphones e *pocket PCs*, também encontrada em veículos para o sistema embarcado de navegação.

Podemos usar também outro sistema operacional embarcado, chamado de *TinyOS*, que está na linha dos sistemas operacionais de softwares livres e permite o uso para resolver problemas relacionados a redes de sensores sem fio, ou seja, aqui entra uma interação com sistema embarcado e hardware que não permite a interferência humana, pois os sensores estarão colocados em equipamentos ou gerenciando uma floresta para saber em que ponto está ocorrendo um incêndio e enviar um alerta para uma equipe de bombeiros resolver esse problema. Tem como facilidade, na sua arquitetura, o emprego de componentes prontos que permitem que a codificação gerada possa ser pequena, além de ajustar a situação de pouca memória que os dispositivos embarcados têm. Como trabalha em rede, o *TinyOS* permite usar uma biblioteca de componentes já configurados para que os sistemas possam interagir com drivers de sensores, protocolos de rede, entre outros serviços de rede.

Figura 1.10 | Exemplo de rede de sensores

Fonte: Teixeira, Almeida e Fikani (2011, p. 11).

Na Figura 1.10, podemos notar a existência de uma rede de sensores que se conecta a um nó sorvedouro, o qual faz a função de um servidor em funcionamento na rede, fazendo com que os dados sejam trafegados entre os sensores, que se autochecam e, também, que trocam informações com o nó sorvedouro. Nesse caso, no computador, existe um sistema operacional embarcado chamado *TinyOS*, que gerencia os nós da rede, permitindo ao usuário verificar o

que pode estar acontecendo com ela naquele momento e, também, exibir relatórios do funcionamento dos sensores da rede sob diversos aspectos, tais como o consumo de bateria.

Ainda podemos citar outra ferramenta para sistemas embarcados, chamada *Contiki*, que foi criada para ser um sistema operacional embarcado e admite total interação com seu código-fonte, pois ele é totalmente aberto, permitindo a aplicação em sistemas multitarefas e com pouca disponibilidade de memória. Sua programação é feita usando a linguagem C e é perfeito para ser usado em microcontroladores, já que o código resultante é pequeno e cabe em dispositivos com pouca capacidade de memória.

Interfaces de sistemas embarcados

Sempre que falamos de sistemas embarcados, já podemos associar que esse tipo de sistema não possui uma forma de inserção de comandos ou configuração, por isso são chamados de embarcados. Para resolver esse problema, o que pode ser feito é deixar uma forma de conexão para que, usando um computador, possamos ter acesso ao sistema operacional embarcado e conseguir executar seus comandos, bem como ter acesso ao *kernel* ou ao sistema que foi embarcado em um microcontrolador, para que possamos operar nesses dispositivos.

Quando uma máquina necessita funcionar por várias horas e repetir os mesmos comandos, trabalhando com ciclos de funcionamento, esse tipo de máquina tem um sistema embarcado que permite o seu funcionamento sem interrupções, o qual pode ser "embarcado" no microcontrolador através de uma interface, que é a comunicação serial com o computador.

Exemplificando

Outra forma de aplicação de sistemas embarcados pode ser encontrada nos postes que fazem suporte para os repetidores de alguns provedores de acesso, que na maioria das aplicações, estão usando sistemas de painéis solares que possuem um sistema embarcado funcionando de forma autônoma para otimizar o uso de energia gerada pelas placas solares.

Esse exemplo pode ser replicado para outros equipamentos, mas estamos falando de uma centrífuga de açúcar para usinas de açúcar e álcool. Essas centrífugas têm que trabalhar em um ciclo de funcionamento, em que devem receber uma matéria-prima, para que, no final, seja produzido o açúcar. Esses ciclos compreendem aceleração e desaceleração de motores, abertura e fechamento de válvulas, entre outras funcionalidades constantes no processo.

Assim, podemos notar a necessidade de um sistema embarcado, que, nesse caso, é colocado em um painel de controle que controlará todo esse processo.

Onde entra a interface (interação com o usuário) com o sistema embarcado? Essa interface se dá por uma entrada existente no painel, que permite a conexão de um notebook para acessar o sistema operacional embarcado e programar os processos do ciclo de funcionamento da centrífuga e, depois, colocar a máquina para trabalhar com os códigos e parâmetros já configurados.

Figura 1.11 | Exemplo de sistema embarcado em centrífuga de açúcar

Fonte: RCA (2010, [s.p.])

Na Figura 1.11, podemos notar que o número 1 se refere ao painel que fica no controle das centrífugas de açúcar de uma linha de produção da usina, e o número dois mostra a *interface* de controle das centrífugas e uma imagem referente ao funcionamento das máquinas.

Outra forma de interface é através de teclados em painéis, que permitem alterar alguns dados já previamente configurados dos sistemas embarcados, mas que, em nenhum momento, são

alterações diretamente no sistema, ou seja, alteramos os dados, mas não o programa, para que seja executada outra configuração ou inserir outro tipo de programa em um equipamento.

Com o intuito de assimilar mais os conceitos sobre sistemas embarcados, assista ao vídeo a seguir, e no tempo que foi estipulado na sua descrição, você poderá verificar os conceitos abordados nesse material. Este vídeo intitula-se: *Introdução aos Sistemas Embarcados*. Disponível em: <https://www.youtube.com/watch?v=1I3QKMzSXUM>. Acesso em: 3 nov. 2017. O aluno deverá assistir de 1:15 até 8:15.

Outra fonte de informação sobre sistemas embarcados pode ser encontrada no vídeo a seguir, em que você verá as aplicações práticas dos sistemas embarcados nos dias de hoje. Disponível em: <https://www.youtube.com/watch?v=FvX33N1JOXY>. Acesso em: 3 nov. 2017.

Dessa forma, podemos introduzir quais são os conceitos e as características mais importantes referentes a sistemas embarcados. Claro que existem muito mais informações em livros, revistas e sites confiáveis na internet, por isso, procure cada vez mais conteúdos referentes a esse tema, que é bem interessante e muito requisitado no mercado nos dias de hoje.

Sem medo de errar

Como podemos notar, nosso cliente tem conhecimento teórico com certa limitação sobre o tema de automação de sua linha de produção, que é o desenvolvimento de um sistema embarcado para que todos os processos que possam ser automatizados sejam realizados. Assim, você foi contratado para fazer este projeto e, até o momento, o desenvolvimento vem sendo satisfatório. Você deverá fazer o levantamento das principais características dos sistemas embarcados, quais são as ferramentas que podemos usar para o desenvolvimento de um sistema e como fazer para “interfacear” o sistema e o usuário final para uso do sistema embarcado que estamos desenvolvendo. Isso deverá ser entregue para a empresa em formato de relatório, no qual cada um dos itens levantados será explicado de

forma a mostrar a importância dos sistemas embarcados nos dias de hoje, principalmente, para o seu cliente. Dessa forma, nosso relatório deverá ser entregue usando padrão ABNT e elencar: as principais características dos sistemas embarcados; quais são as categorias dos seus componentes; as ferramentas de desenvolvimento; e como podemos fazer a interface para sistemas embarcados.

Feito isso, a resolução poderá se encontrar da seguinte forma:

Uma capa com o título: "Componentes dos sistemas embarcados" e deverá conter também o nome dos proprietários da empresa consultada (você, aluno).

Ainda pensando nas normas ABNT, devemos colocar um Sumário para mostrar quais são os itens abordados no relatório. Note que, aqui, você poderá fazer após o término do relatório ou inserir um Sumário automático, que é mais interessante.

A parte mais importante é o conteúdo do relatório. Podemos começar explicando brevemente o que um sistema embarcado pode fazer e, depois, elencar suas principais características, como a limitação de memória e bateria, no caso de um sistema ter que ficar em local remoto.

Assim que finalizar esse item, podemos partir para os componentes relacionados aos sistemas embarcados. Listamos os componentes por ordem de funcionalidade ou importância, fica a critério, mas temos que colocar os principais, que são a Unidade Central de Processamento, que é o componente mais importante de um sistema embarcado, enfatizando que nela são feitos todos os cálculos e os processamentos necessários para o bom funcionamento do sistema; depois, os sensores e os atuadores, que atuam como peças fundamentais na entrada de dados e na execução dos resultados gerados pela Unidade Central de Processamento.

A partir desse ponto, nosso relatório poderá abordar quais ferramentas podem ser usadas para que um projeto de sistema embarcado possa ser executado com êxito e, então, podemos citar as duas linhas mais usadas no mercado, que são as ferramentas proprietárias e as ferramentas livres.

Por fim, podemos mostrar para o nosso cliente, no relatório, quais são as formas de interface com sistemas embarcados que são utilizadas e como fazer esse tipo de interface; o que, necessariamente,

queremos com essa interface e o que podemos fazer quando acessamos um sistema embarcado diretamente no microcontrolador dentro de um equipamento. Os sistemas embarcados, depois de colocados em produção, deverão receber o mínimo de interferência humana para que os processos sejam executados a contento.

Avançando na prática

Projetando um sistema embarcado para receptor de TV

Descrição da situação-problema

A TV “por assinatura”, “a cabo”, entre outras denominações possíveis, é um serviço de acesso ao conteúdo de canais de televisão abertos e dos canais “fechados” ou pagos. O serviço de TV “por assinatura” utiliza também a transmissão do sinal por tecnologia digital. No entanto, esse sinal chega à residência do telespectador por meio de uma antena parabólica especial ou de um cabo conectado ao receptor, que, por sua vez, é conectado ao televisor.

Você é convidado por uma empresa de TV "por assinatura", que está iniciando suas atividades em sua cidade, a realizar um projeto de sistema embarcado para ser utilizado em aparelhos receptores de sinal digital. Dessa forma, você e sua equipe deverão realizar o levantamento do software e das ferramentas para a construção de sistema embarcado. Como produto final, você deve entregar um relatório, “Projeto Receptor de TV”, que contenha todos os elementos de software e as ferramentas para a construção desse sistema embarcado.

Resolução da situação-problema

Para a construção desse relatório, você deve realizar as seguintes atividades:

- Levantamento das principais marcas e modelos de receptores de TV digital disponíveis e que possibilitem serem programados.

- Análise dos principais sistemas operacionais para a construção de sistemas operacionais, levantando as vantagens e as desvantagens de cada ferramenta.

- Proposta dos componentes que serão utilizados no sistema embarcado.

Faça valer a pena

1. Quando existe a necessidade de projeto de um sistema embarcado, devemos pensar em como os conceitos básicos podem ser úteis no momento de conversar com um cliente ou com um funcionário que desenvolverá a parte física do sistema.

Qual é a definição de unidade de processamento para sistemas embarcados?

a) A unidade de processamento serve para processar os dados recebidos pelos atuadores e mostrar os resultados para o usuário.

b) A unidade de processamento permite que o usuário digite os dados do sistema embarcado e mostre os resultados na tela.

c) A unidade de processamento é um componente de um sistema embarcado que não tem muita importância para o usuário.

d) A unidade de processamento é responsável pela execução de todos os cálculos e pela exibição dos resultados para o usuário e para os atuadores.

e) A unidade de processamento permite o processamento de cálculos matemáticos e também de dados importantes para o sistema.

2. Quando se fala de ferramentas para o desenvolvimento de sistemas embarcados, temos que lembrar das ferramentas proprietárias e livres. Muitas pessoas preferem as proprietárias, e outras, as ferramentas livres. Isso se deve ao fato de como poderá ser o suporte quanto à necessidade de esclarecimentos referentes ao seu processo de funcionamento e desenvolvimento.

Dessa forma, como podemos escolher entre os dois tipos de paradigmas com relação às ferramentas de desenvolvimento para sistemas embarcados? Por quê?

a) Dependerá da escolha da empresa, já que devemos fazer o projeto conforme a empresa necessitar e não opinar sobre a decisão.

b) Dependerá do cliente e do projeto, porque, em muitas ocasiões, os requisitos do sistema definirão qual tipo de ferramenta escolher.

c) Todos os projetos devem ser feitos usando as duas ferramentas integradas, para que o custo seja dividido e seja cobrado menos.

d) Dependerá se o cliente gosta de ferramentas proprietárias ou de ferramentas livres.

e) Cada projeto deve ser analisado individualmente, assim, devemos indicar somente as ferramentas livres.

3. As características de sistemas embarcados têm como previsão ser multitarefas, para que problemas específicos sejam resolvidos de forma que os processos sejam executados de forma concorrente. Assim, podemos dizer que os sistemas embarcados não devem ser usados nos problemas mais genéricos que podem ser resolvidos.

Para um sistema embarcado multiprocessado, ele pode ser desenvolvido usando a capacidade de ser multiprogramável?

a) Sim, pois um sistema embarcado tem uma grande quantidade de memória para processamento de dados.

b) Não, pois um sistema embarcado pode processar uma grande quantidade de dados e pode ser multiprogramado.

c) Não, pelo fato de ter somente um microcontrolador que tem pequena capacidade de processamento.

d) Sim, porque um sistema embarcado tem a necessidade de ser multiprocessado e ter uma grande quantidade de processamento.

e) Sistemas embarcados não têm como características importantes o uso de memória e processamento.

Seção 1.3

Hardware para sistemas embarcados

Diálogo aberto

Olá, aluno! Daremos sequência aos nossos estudos falando sobre hardware para sistemas embarcados. Este tema é interessante, pois nos levará a mostrar para o nosso cliente quais são os tipos de hardwares disponíveis para trabalhar com sistemas embarcados e como aplicá-los no sistema embarcado que será adotado pela empresa, com o objetivo de automatizar todos os processos que forem necessários. Também mostraremos para a empresa quais os elementos das ferramentas para sistemas embarcados e como fazer a aplicação dessa ferramenta no sistema que será de propriedade da empresa. Para isso, você deverá, ao final desta unidade, mostrar para as pessoas diretamente interessadas no projeto de automação como foi feita a escolha do hardware para o sistema embarcado e as ferramentas que serão usadas para o desenvolvimento desse hardware.

Na necessidade de um projeto de sistema embarcado, você precisa saber como os componentes de hardware podem influenciar na performance do sistema, bem como em um possível levantamento de custo, qual seria o impacto sobre o projeto. Por exemplo, este possível projeto tem que ter a escolha de um hardware com uma plataforma de prototipação, como o Arduino, mais um sistema para desenvolvimento do software (esta será nossa ferramenta) e outro programa que permite o desenvolvimento do projeto eletrônico para o sistema embarcado.

Então, no final da unidade, será entregue um relatório que permita aos usuários da empresa entender qual hardware será utilizado, como ele será programado e com que ferramenta ele será prototipado.

Não pode faltar

Nossa atenção estará voltada ao projeto de hardware para sistemas embarcados. Hardwares são os componentes físicos dos produtos eletrônicos ou que têm algum tipo de automação e que

permitem que o software embarcado dentro deles execute uma determinada função.

De acordo com Carro e Wagner (2003), um projeto de hardware para sistemas embarcados consiste em encontrar um meio derivado de um projeto para que possa atender aos requisitos de uma aplicação específica, como o desempenho do hardware e o consumo de energia, que é particionado entre o projeto de hardware e software. Por exemplo, podemos citar que a arquitetura de hardware de um smartphone e de um sistema de freio ABS tem muita similaridade, mas com certeza o software que controla esses sistemas é completamente diferente.

Os sistemas embarcados são compostos por componentes de hardware e software que, nos dias atuais, são comuns na maioria dos projetos. Dessa forma, quando há a necessidade de levantamento de requisitos para um sistema embarcado que tenha grande similaridade com outro sistema já projetado, tende-se a fazer reúso dessas plataformas e componentes para que o tempo de desenvolvimento do projeto possa ser abreviado sem que se tenha prejuízo quanto à qualidade na entrega do produto.

Então, podemos entender o reúso em sistemas embarcados como sendo o processo de usar partes de um projeto que já está em funcionamento, para que seja reaproveitado no projeto atual. Esse é um dos conceitos da engenharia de software que é utilizado para sistemas embarcados.

Essa é uma tendência no mundo de hoje, pois com a concorrência cada vez mais acirrada, as empresas tendem a procurar formas de acelerar o processo de desenvolvimento de seus produtos, em qualquer área.

Na área de sistemas embarcados não é diferente, pois os componentes de hardware que são utilizados para um determinado projeto podem ser reusados no desenvolvimento de outro projeto. Por exemplo, os componentes microcontroladores, entre outros, que um projeto tem e que com certeza serão usados em outro projeto, não tem a necessidade de um novo levantamento de requisitos, faz-se o reúso desses requisitos para o projeto novo e levantam-se os requisitos que serão diferentes do primeiro.

Partindo desse pressuposto, um projeto de hardware para sistemas embarcados deverá integrar os componentes físicos, que

são os microcontroladores, os sensores e os atuadores, com o sistema computacional embarcado dentro de microcontroladores e microprocessadores, para obter um resultado como resposta a um determinado problema.

Isso é possível com algumas técnicas que permitem separar o projeto de hardware do projeto de software e pensar em reúso de artefatos gerados para a construção de um projeto em novos desafios relacionados a sistemas embarcados.

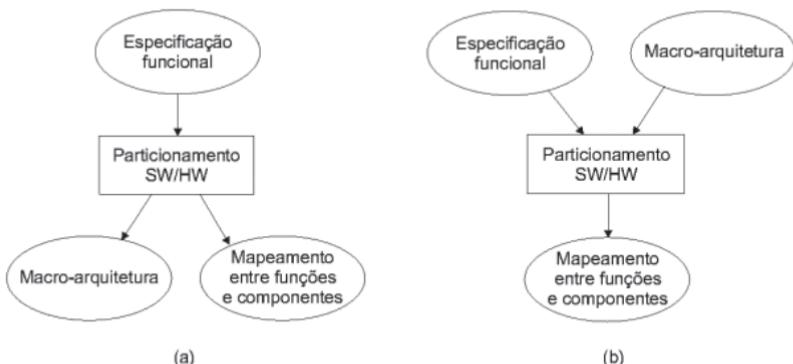
Podemos, então, partindo de uma metodologia que permita ter precisão no levantamento dos dados para o projeto, escolher os componentes que farão parte do sistema embarcado partindo dos seguintes pressupostos, segundo Zurita (2014, p. 5):

1 - Permitir que o cumprimento dos requisitos do projeto possa ser assegurado de maneira formal à medida que o projeto avança.
2 - Potencializar o uso de ferramentas de automação das tarefas, pois cada etapa conhecida pode ser dividida em tarefas menores.
3 - Facilitar a comunicação entre as equipes de desenvolvimento, já que o estabelecimento da metodologia também auxilia na divisão das responsabilidades, permitindo que cada equipe tome ciência das atribuições das demais e da maneira como estão vinculadas ao projeto.

Assim, com uma metodologia de projeto adotada, pode-se garantir o cumprimento de três ações que todo projeto possui, que são as seguintes:

Especificação e modelagem do sistema embarcado, que tem início quando partimos de uma especificação para o sistema e termina com a descrição de um modelo claro do sistema. Essa especificação deve ser funcional, ou seja, tem um conjunto de relações que envolvem entradas, saídas e possíveis estados internos do sistema, podendo essas relações serem implícitas ou explícitas.

Reflita

Para sua reflexão, quando falamos em desenvolvimento do projeto de sistemas embarcados, partimos de um conjunto de requisitos especificados de forma genérica para que o projetista faça todo o processo de criação

do modelo de sistema embarcado. O que aconteceria se este projeto fosse iniciado sem o levantamento desses requisitos? Haveria algum tipo de prejuízo para o desenvolvimento do projeto?

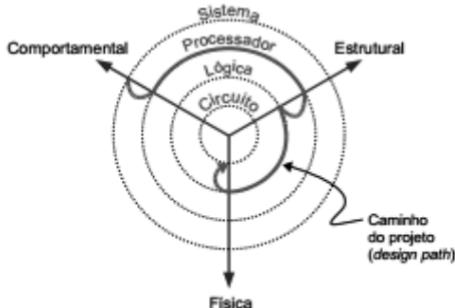
Após, levantar um conjunto de propriedades que o projeto deve satisfazer, dadas na forma de um conjunto de relações de entrada, saídas e estados do sistema que devem ser comparados com a especificação funcional.

Devemos prever um conjunto de índices que possam ser usados para medir o desempenho da qualidade do projeto em termos de quanto está custando, qual o grau de confiabilidade, a velocidade de desenvolvimento, e tudo em forma de equações, que permitem dados de entrada e saída do sistema. Outro ponto importante é que temos que pensar em um conjunto de restrições sobre os índices de desempenho, já que o projeto é um modelo e todo modelo consiste em pontos de verificação e também tem restrições.

Segundo Carro e Wagner (2003), a primeira etapa consiste no particionamento entre hardware e software, que tem como objetivo, a partir da especificação funcional dos requisitos do sistema, saber quais descrições serão relacionadas ao desenvolvimento do hardware e do software e que gerará como saída o mapeamento de cada função da especificação e uma descrição de um componente da arquitetura, que pode ser um processador ou um bloco que representa um componente de hardware.

Figura 1.12 | Particionamento e macroarquitetura

Fonte: adaptada de Carro e Wagner (2003, [s.p.]).

Na Figura 1.12, podemos entender como este particionamento se dá. Olhando a letra (a), vemos que uma macroarquitetura, que é uma descrição macro do que é necessário para a arquitetura do projeto de sistema embarcado, deveria ser gerada a partir de um processo de particionamento automático, que tem início com um conjunto de especificações funcionais. Isso do ponto de vista computacional se torna bastante complexo, tanto que o normal é, a partir de uma macroarquitetura já definida, particionar os componentes de hardware e software para encontrar as funções e os componentes de um sistema, conforme mostrado na letra (b).

O que precisamos entender é que, a partir de um conjunto de especificações funcionais, nosso projeto de hardware pode ser transformado em um conjunto de macrofunções, que permitirá escolher quais tipos de hardware deveremos usar.

Após modelado o sistema e feito o particionamento das macrofunções, devemos partir para o processo de validação do modelo de hardware gerado e, para isso, usamos duas ferramentas nesta etapa, que são a simulação e a verificação formal. Como todo processo de validação, no caso de encontrar algum erro, todo o processo deve ser repetido até que o modelo seja totalmente validado.

De acordo com Zurita (2014), depois de validado o modelo, podemos passar para o processo de síntese, para que a partir de entradas, como um software, termos como saída a descrição do hardware, que será usado no sistema embarcado, ou termos como entrada a especificação do sistema, através da descrição do hardware, e ter como saída uma descrição de hardware mais detalhada.

Nesse ponto, podemos adotar várias metodologias para o desenvolvimento e para sabermos os níveis de abstração de um projeto de sistema embarcado. Uma das formas mais utilizadas pelos projetistas é o uso de um diagrama chamado de Diagrama em Y, criado por Gajski, e que é usado para representar a informação a respeito de um sistema. A Figura 1.13 mostra esse diagrama e suas características.

Figura 1.13 | Diagrama em Y de Gajski

Fonte: Zurita (2014, p. 6).

Nota-se que o diagrama da Figura 1.13 consiste em três dimensões, divididas em estrutural, comportamental e física. Na dimensão estrutural, representa-se um conjunto de componentes e conexões; na dimensão comportamental, cada componente do sistema é visto como uma caixa-preta, em que as saídas serão descritas em termos de entradas aplicadas em função do tempo, ou seja, nesta camada, serão descritos o que são os componentes, mas não como esses componentes serão. Por fim, na dimensão física, teremos as informações sobre as características físicas dos componentes.

Esse diagrama pode ser interpretado de fora para dentro, em que podemos verificar círculos concêntricos que permitem aumentar ou diminuir os níveis de detalhamento do sistema embarcado. Assim, quando estamos falando da parte de fora, temos um nível de detalhamento menor e, conforme caminhamos para dentro, a complexidade aumenta.

Sobre as metodologias de desenvolvimento, anteriormente, estudamos os níveis de abstração para se chegar a um modelo de sistema, assim, podemos usar, nos dias atuais, três categorias para um projeto, que são:

Bottom-up - permite o seu início a partir de uma descrição detalhada dos componentes principais de um sistema. A partir daí, todos os componentes serão conectados a subsistemas, os quais serão conectados a subsistemas maiores, e assim por diante, até que o sistema seja completado.

Nesse modelo, os projetistas desenvolvem uma série de componentes que serão armazenados em bibliotecas, para que cada

nível de abstração possa ser separado, a fim de ter mais controle no momento de gerenciamento do projeto.

Top-down - a partir de características finais do sistema e sem detalhamento, o projeto de inicia e vai se detalhando cada vez mais e se refinando, sendo dividido em subsistemas, que serão divididos em subsistemas menores até se chegar no nível de componentes do sistema. É uma metodologia intuitiva, com a vantagem de se automatizar grande parte do processo, fazendo com que se ganhe em redução do tempo de projeto e ajude a minimizar falhas humanas.

Meet-in-the-middle - combina as duas metodologias anteriores para que se possa aproveitar as vantagens de cada uma delas, e tenta permitir que as desvantagens sejam evitadas. O projeto aqui se inicia com uma abstração, que vai ser refinada sucessivamente para níveis cada vez mais baixos até o nível intermediário, em que são gerados os componentes em uma biblioteca de componentes. Assim, cada componente pode ser descrito em termos de descrição de abstrações feitas através de técnicas da metodologia *bottom-up*.

Para que possamos entender os elementos que compõem os sistemas embarcados, devemos ter em evidência como é o ciclo de vida do projeto para sistemas embarcados. Este ciclo de vida, segundo Zurita (2014, p. 7), está dividido em sete fases:

Especificação do produto.
Particionamento do projeto em componentes de hardware e software.
Iteração e refinamento do particionamento.
Tarefas independentes de projeto do hardware e software.
Integração dos componentes de hardware e software.
Testes, aceitação e lançamento do produto.
Manutenção e atualização contínua.

A primeira fase consiste em saber como o sistema se comportará a partir de uma série de especificações descritas de maneira informal. Na segunda fase, os projetistas devem se ater à definição de quais partes devem ser desenvolvidas em hardware e quais partes são desenvolvidas em software, que podem ser tanto blocos de funções definidas nas especificações quanto podem ser divididas em sub-blocos dessas funcionalidades.

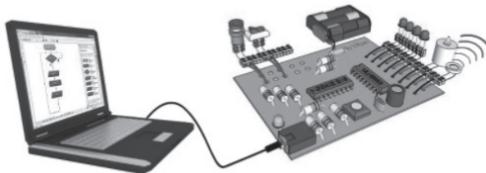
Na terceira fase, com o uso de ferramentas de alto nível, como ferramentas de *benchmark*, que permitem testar como um componente se comporta tanto em hardware como em software, serão feitos os testes dos particionamentos realizados na fase anterior.

Na quarta fase, serão executados procedimentos para que se possam encontrar erros de desenvolvimento de hardware e software e terá como objetivo principal corrigir esses erros, de forma que, após essas correções, esses componentes possam ser reunidos e integrados para compor o sistema embarcado de nosso interesse.

A quinta fase tem como característica integrar os dois componentes do sistema embarcado para que funcionem de forma organizada e que atendam às entradas de dados específicas e retornem os dados especificados com saída.

Na sexta fase, deveremos nos atentar aos testes, os quais farão com que os usuários finais possam interagir com o sistema e, depois disso, fazer um questionamento para eles, a fim de verificar qual foi o nível de aceitação do produto, para que, em projetos futuros, possamos corrigir algumas falhas encontradas somente no ambiente de produção.

A última fase é uma das mais importantes e ignorada por grande parte das empresas prestadoras de serviço, que é a fase de manutenção e atualização do sistema desenvolvido e em uso. Nessa fase, deve-se resolver os problemas que possam acontecer com o sistema, bem como otimizar as características do produto e entregar a documentação deste sistema para que tanto os clientes externos quanto os clientes internos possam saber quais são todas as funcionalidades e as características do sistema.

Assim, poderemos falar, agora, das ferramentas que podem ser usadas em desenvolvimento de sistemas embarcados. Algumas delas são de uso comum por usuários e profissionais da área de sistemas embarcados; outras são mais específicas para empresas de desenvolvimento de produtos em larga escala.

Ambientes de desenvolvimento podem ser chamados também de Ambiente Integrado de Desenvolvimento, que vem da sigla em inglês IDE, que significa: *Integrated Development Environment*, ou Ambiente de Desenvolvimento Integrado. Quando falamos de sistemas embarcados, podemos ter o Proteus, da empresa Labcenter Electronics Ltda., que

permite simular circuitos eletrônicos para sistemas embarcados e gravar estes sistemas através de um kit de gravação para microcontroladores PIC, entre outros.

Uma das ferramentas usadas para esse tipo de projeto se chama Circuit Wizard, que tem uma interface gráfica, na qual é permitido criar o esquema dos componentes de hardware e software que serão usados no projeto do sistema embarcado e um kit que permite transferir o sistema simulado no computador para um microcontrolador que fará parte do sistema. Na Figura 1.14, podemos ver essa ferramenta em funcionamento.

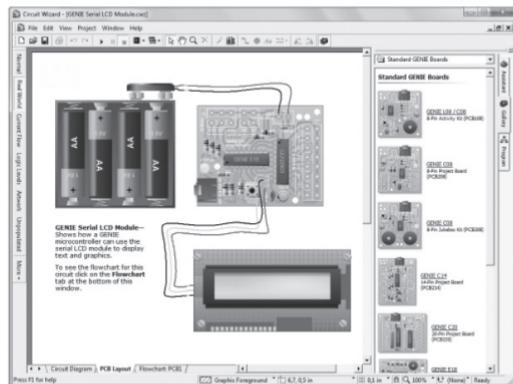
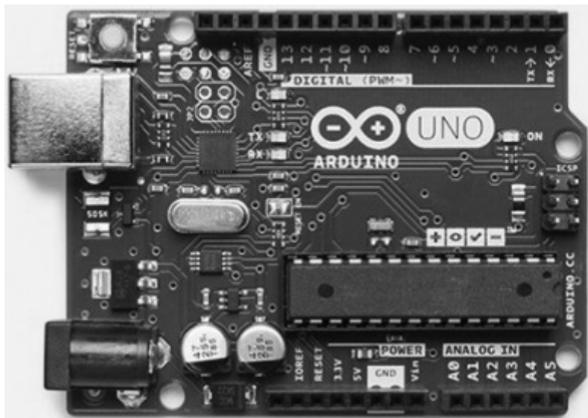



Figura 1.14 | Kit de desenvolvimento do Circuit Wizard

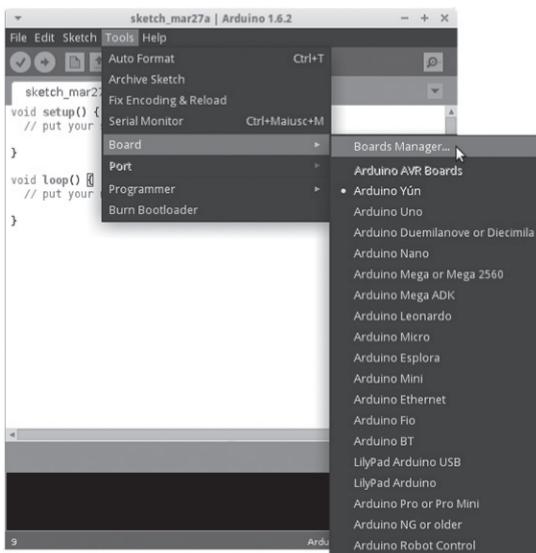
Fonte: Soares (2013, [s.p.])

Esse kit permite o desenvolvimento de sistemas embarcados usando o microcontrolador PIC de todas as categorias. Na tela do computador, pode-se verificar o fluxograma de funcionamento do sistema, bem como existe a possibilidade de criar o esquema com os componentes que serão usados e testá-lo antes de gravar o sistema no microcontrolador, conforme mostra a Figura 1.15.

Figura 1.15 | Esquema simulando no software do Circuit Wizard

Fonte: Soares (2013, [s.p.])

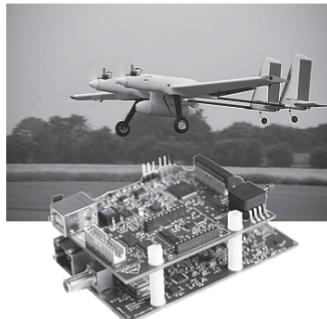
Outra ferramenta de desenvolvimento de sistemas embarcados que pode ser utilizada na criação de projetos é a plataforma Arduino, que tem como principal objetivo, desde sua criação, desenvolver produtos de hardware e software embarcados a partir de protótipos antes de criar o produto em larga escala.

Mostraremos, na Figura 1.16, como é o hardware do Arduino e, na Figura 1.17, o ambiente de desenvolvimento do software para o sistema embarcado.

Figura 1.16 | Placa de hardware Arduino UNO

Fonte: Arduino (2017, [s.p.])

Podemos notar nesta placa que existem vários componentes já integrados, os quais permitem o rápido desenvolvimento e o teste de circuitos e projetos para sistemas embarcados, já que a ideia principal dos criadores do Arduino foi esta, evitar o gasto desnecessário de tempo para o projeto de sistemas embarcados.

Ainda, nota-se que essa placa tem portas de entrada analógicas e digitais para que qualquer tipo de configuração possa ser implementado. Dessa forma, a aceitação do mercado foi tamanha que as placas Arduino começaram a fazer parte dos projetos desenvolvidos com essa plataforma.

Figura 1.17 | Ambiente de desenvolvimento para Arduino

Fonte: Arduino (2017, [s.p.])

Na Figura 1.17, podemos notar que o ambiente de desenvolvimento do Arduino é bem simples e com fácil navegabilidade, pois para se usar esse ambiente de programação, necessita-se apenas saber a linguagem C, escolher qual porta de comunicação será usada e, por fim, ligar o computador à placa através de um cabo USB.

Exemplificando

Falamos no texto sobre duas ferramentas de desenvolvimento para sistemas embarcados. Outra ferramenta muito utilizada é o ambiente integrado de desenvolvimento chamado Proteus. Veja como podemos usar essa ferramenta no exemplo a seguir.

Este é um projeto de um sistema embarcado para gerenciamento de um avião não tripulado, ou como a empresa LabCenter mostra o Unmanned Aircraft, que foi desenvolvido usando a sua IDE Proteus para desenvolver tanto o software quanto o hardware para o avião.

Com essas duas ferramentas, podemos fazer o projeto lógico e físico, os testes, a integração do hardware e do software com grande liberdade, evitando gastar muito tempo de desenvolvimento.

Para maiores informações sobre projetos de sistemas embarcados, você poderá acessar os seguintes vídeos:

Sensor de estacionamento com Arduino. Disponível em: <https://www.youtube.com/watch?v=NJUroL_oTo8>. Acesso em: 10 nov. 2017.

Projeto de um Sistema Embarcado com Microcontrolador AVR. Disponível em: <https://www.youtube.com/watch?v=6c-Kt84R7Ds&t=165s>. Acesso em: 27 dez. 2017.

Projeto de sistema embarcado para uma máquina de lavar. Disponível em: <https://www.youtube.com/watch?v=b5y2R0djoDQ>. Acesso em: 10 nov. 2017.

Projeto de um sistema embarcado para detecção de presença com Arduino. Disponível em: <https://www.youtube.com/watch?v=o66BOLcjnJs>. Acesso em: 10 nov. 2017.

Agora é com você, vamos em frente.

Sem medo de errar

Relembrando nossa situação-problema, foi solicitado um relatório com a listagem dos componentes de hardware disponíveis para trabalhar com o sistema embarcado que será adotado pela empresa para automatizar seus processos. Será necessário também apresentar para a empresa quais são as ferramentas para o sistema embarcado e como fazer a aplicação dessa ferramenta no sistema, que será de propriedade da empresa. Para isso, você deverá, no final desta unidade, mostrar para as pessoas diretamente interessadas no projeto de automação como foi feita a escolha

do hardware para o sistema embarcado e quais ferramentas que serão usadas com esse hardware.

Na necessidade de um projeto de sistema embarcado, você precisa saber como os componentes de hardware podem influenciar na performance do sistema, bem como seu custo e o impacto sobre o projeto.

Para a sua solução, a equipe deve realizar o levantamento do hardware disponível para a criação de sistemas embarcados, optando pelo uso do Arduino (hardware) e/ou um sistema para desenvolvimento do software (esta será nossa ferramenta), como a linguagem C/C++.

Para isso, nosso relatório será apresentado da seguinte forma:

- Uma capa, contendo a descrição do que é o relatório.

- Um sumário.

- Desenvolvimento do tema: aqui, você deverá mostrar para a empresa qual o processo de escolha do projeto e sua integração com os hardwares para sistemas embarcados; depois, como podem ser aplicados esses hardwares para sistemas embarcados no projeto de automação da empresa, mostrando quais elementos das ferramentas desse tipo de sistema serão usados no projeto de automação e como aplicar essas ferramentas no projeto.

- Conclua, mostrando todo o processo de escolha dos hardwares e ferramentas para sistemas embarcados que você propõe para a empresa.

- Liste as referências bibliográficas usadas no seu relatório.

Tenha em mente que esse relatório será mais um documento com os requisitos levantados no início de nosso livro didático, que permitirá saber como um projeto para sistemas embarcados pode ser desenvolvido e quais tecnologias serão usadas para que esse desenvolvimento ocorra.

Na prática, os profissionais entregam relatórios, projetos e produtos prontos para que os possíveis envolvidos no desenvolvimento dos projetos possam saber como os processos ocorrerão e o que usar.

Avançando na prática

Projeto de sistema embarcado com Arduino - medindo temperatura

Descrição da situação-problema

Uma empresa solicitou um projeto de sistema embarcado para um de seus equipamentos que está aquecendo demasiadamente. Este equipamento é de grande importância no processo de fabricação da empresa, mas não tem nenhum controle de temperatura instalado nele. Dessa forma, quando a temperatura excede a um limite de 50 °C, o equipamento começar a falhar e desliga. A proposta é fazer um sistema de controle de temperatura que, quando a temperatura aumentar repentinamente por causa do aquecimento gerado por seu trabalho, faça com que seja acionado um sistema de ventilação, que terá como objetivo deixar a temperatura menor que os 50 °C estabelecidos pelo fabricante do equipamento, e esse projeto tem como premissa ser de baixo custo de desenvolvimento.

Resolução da situação-problema

Para se fazer um projeto com baixo custo de desenvolvimento, podemos escolher a plataforma Arduino. Primeiramente, porque tem um baixo custo, basta fazer uma pequena pesquisa sobre uma placa Arduino UNO para que esse requisito seja sanado.

Depois, os sensores de temperatura que atuam com o Arduino são bem acessíveis e encontrados em qualquer casa de equipamentos para eletrônica. Por fim, necessitamos de uma IDE para o desenvolvimento da aplicação, no caso, será usada a IDE da placa Arduino, por se tratar de um programa sem custo e que permite transferir os programas diretamente para a placa através de um cabo USB.

Para que nosso código seja criado de forma correta, poderemos pesquisar nos seguintes endereços, que nos darão uma ideia do que poderemos programar para esse sistema de controle de temperatura. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/3252633/mod_resource/content/1/Guia_Arduino_Iniciante_Multilogica_Shop.pdf>; <http://apostilas.eletrogate.com/Apostila_Arduino_Basico-V1.0-Eletrogate.pdf>. Acesso em: 10 nov. 2017.

Por fim, necessitaremos de um atuador, que ligará um sistema de ventilação composto por um ventilador instalado próximo ao local do equipamento que mais aquece e que gera a falha no seu funcionamento. Portanto, será entregue ao cliente:

1- Uma lista com todos os componentes eletrônicos necessários para o projeto.

2- Um esquema que permite ver as ligações dos componentes de hardware.

3- Uma listagem do código-fonte criado para o projeto do sistema de controle de temperatura.

4- Uma mostra de quanto custará o projeto com relação aos componentes eletrônicos.

Isso deve ser entregue em documento Word, com todas as premissas da norma ABNT necessárias.

Faça valer a pena

1. Quando precisamos fazer um projeto de sistemas embarcados, temos a necessidade de obter uma descrição detalhada das características do que deverá ser projetado para o sistema. Assim, o projeto de um sistema embarcado está baseado em um ciclo de vida que compreende sete fases.

Qual fase do ciclo de vida permite que o projeto de software possa ser testado com o projeto de hardware?

a) Fase de concepção do projeto de sistemas embarcados.

b) Fase de integração dos componentes de hardware e software.

c) Fase de separação dos componentes de hardware e software.

d) Fase de validação do sistema embarcado.

e) Fase de testes com o usuário do sistema embarcado.

2. Zurita (2014) mostra os pressupostos para o desenvolvimento de projetos de sistemas embarcados que permitem nos dar um norte de como podemos começar a pensar no que devemos ter como garantia nos projetos. Esses pressupostos estão dispostos a seguir:

1 – Permitir que o cumprimento dos ____________ do ____________ possa ser assegurado de maneira formal à medida que o projeto avança.

2 – Potencializar o uso de _______________ de automação das tarefas, pois cada etapa conhecida pode ser dividida em tarefas menores.

3 – Facilitar a ________________ entre as equipes de desenvolvimento, já que o estabelecimento da _______________ também auxilia na divisão das _____________________, permitindo que cada equipe tome ciência das atribuições das demais e da maneira como estão vinculadas ao projeto.

Preencha as lacunas que foram deixadas no texto:

a) Projeto, ferramentas, requisitos, metodologia, comunicação, responsabilidades.

b) Comunicação, ferramentas, requisitos, projeto, comunicação, responsabilidades, metodologia.

c) Comunicação, projeto, ferramentas, comunicação, responsabilidades, metodologia.

d) Ferramentas, projeto, requisitos, comunicação, responsabilidades, metodologia.

e) Requisitos, projeto, ferramentas, comunicação, metodologia, responsabilidades.

3. Quando existe a necessidade de se trabalhar com projetos de sistemas embarcados, precisamos de algumas ferramentas, as quais, geralmente, são um ambiente de desenvolvimento para o hardware, um programa de computador que permite a programação da parte lógica do sistema embarcado e um ambiente que permite integrar essas duas partes.

Dessa forma, quando precisamos deste programa que nos ajudará a criar os códigos e a lógica de um projeto de sistema embarcado, existe, hoje em dia, uma categoria de programas que permite a criação do código e também integrar com o hardware para o sistema embarcado. Como se chama esse tipo de ferramenta?

a) *RDE - Related Development Equipament.*

b) *EDI - Environment, Development Integrated.*

c) *IDI - Integrated Development Integrated.*

d) *IDE - Ambiente de Desenvolvimento Integrado.*

e) *CDI - Component Development Integrated.*

Referências

ARDUINO. **Arduino**. 2017. Disponível em: <https://www.arduino.cc/>. Acesso em: 2 set. 2017

CARRO, L.; WAGNER, F. R. **Sistemas computacionais embarcados**. 2003. Disponível em: <ftp://ftp.inf.ufrgs.br/pub/flavio/cmp231/jai2003.pdf>. Acesso em: 30 set. 2017.

CISCO. **Roteadores da série 1700**. 2017, Disponível em: <https://www.cisco.com/c/en/us/obsolete/routers/cisco-1700-series-modular-access-routers.html>. Acesso em: 25 set. 2017.

DELAI, A. L. **Sistemas embarcados**: a computação invisível. 2013. Disponível em: <http://www.hardware.com.br/artigos/sistemas-embarcados-computacao-invisivel/conceito.html>. Acesso em: 1º set. 2017.

EQUIPE ARARIBOIA. **Divulgação de fontes alternativas de energia**. 2017. Disponível em: <http://www.equipeararIboia.uff.br/o-projeto/>. Acesso em: 2 out. 2017.

OLIVEIRA, S. de. **Internet das coisas com ESP8266**, **Arduino** e **Raspberry Pi**. São Paulo: Novatec, 2017.

PRADO, S. **Sistemas de tempo real**. 2010. Disponível em: <https://sergioprado.org/sistemas-de-tempo-real-part-1/>. Acesso em: 1º set. 2017.

PUHLMANN, H. **Sistemas operacionais de tempo real**: introdução. 2014. Disponível em: <https://www.embarcados.com.br/sistemas-operacionais-de-tempo-real-rtos/>. Acesso em: 25 set. 2017.

RCA. **RCA automação**. 2010. Disponível em: <http://www.rcaautomacao.com.br/AcucarAlcool1.php>. Acesso em: 25 set. 2017.

ROSSI, H. **Arquitetura de software em sistemas embarcados**. 2015. Disponível em: <https://www.embarcados.com.br/arquitetura-de-software-em-sistemas-embarcados/>. Acesso em: 2 set. 2017.

SOARES, Nelson Vicente. **Circuit Wizard**: Um novo e revolucionário simulador. 2013. Disponível em: <http://blog.render.com.br/eletronica/circuit-wizard-um-novo-e-revolucionario-simulador/>. Acesso em: 2 set. 2017.

TEIXEIRA, R.; ALMEIDA, K.; FIKANI, A. **Redes de sensores**. 2011. Disponível em: <https://pt.slideshare.net/ricardoteix/redes-de-sensores>. Acesso em: 25 set. 2017.

ZURITA, M. E. P. **Projeto de sistemas embarcados**. 2014. Disponível em: <https://www.researchgate.net/publication/267298521>. Acesso em: 5 out. 2017.

Sistemas operacionais embarcados e manipulação de dispositivos

Convite ao estudo

Prezado estudante, nesta unidade, serão apresentados os elementos iniciais de um sistema operacional embarcado. Com base nestes elementos, você terá informações necessárias para a tomada de decisão em projetos e ainda argumentos para a escolha neste tipo de projeto.

O sistema operacional é o nível de software base de um computador ou sistema embarcado, ele fornece uma camada para que a aplicação ou usuário não tenha a necessidade de entender todos os detalhes de funcionamento do computador. O estudo desta unidade nos permitirá resolver o problema da empresa de sistemas embarcados EmbarconEDU, que está no mercado há mais de 15 anos, criando diversas soluções para empresas de segurança. Seu mais novo projeto consiste no desenvolvimento de um sistema embarcado de monitoramento, possuindo sensores infravermelhos, câmeras e atuadores. Com esse hardware será possível reconhecer faces e até detectar para qual direção um possível invasor está olhando.

É importante que esses elementos de captura tenham sistemas embarcados para garantir a robustez do sistema, bem como um curto tempo de reposta. Além das escolhas sobre qual hardware utilizar, o que gera grande preocupação é a escolha do sistema operacional a ser utilizado no projeto, principalmente devido ao curto tempo de desenvolvimento que foi destinado à execução do projeto.

Durante as seções desta unidade, você conhecerá os tipos de sistemas operacionais embarcados, bem como suas características e quais as melhores aplicações para cada tipo.

Tal conhecimento lhe permitirá tomar decisões acertadas em um projeto embarcado, garantindo o sucesso do seu produto.

Esse é um ótimo momento para ter uma visão crítica sobre quais escolhas devem ser tomadas no início de um projeto. O desenvolvimento de um produto com software embarcado necessita de empenho, pois, através das possibilidades de depuração e correção do código, possibilitará a garantia de qualidade. Essa postura de persistência é essencial para assegurar o bom aproveitamento. Nesta etapa, serão apresentados:

- Definição de sistema operacional embarcado.
- Ferramentas e bibliotecas do SO embarcados.
- Introdução e aplicações de SO embarcados de tempo real.
- Visão geral do desenvolvimento de sistema embarcado com SO.

Seção 2.1

Sistemas operacionais embarcados

Diálogo aberto

Caro aluno, iniciaremos agora o estudo sobre os sistemas operacionais embarcados. Conheceremos os componentes básicos, bem como entenderemos as principais diferenças entre um sistema operacional de uso geral e um voltado para circuitos embarcados. Ao final desta seção, você será capaz de identificar os elementos iniciais necessários ao desenvolvimento de produtos com um sistema operacional embarcado.

Para mediar nosso aprendizado, começaremos a resolver o problema da EmbarconEDU, que deseja desenvolver um sistema embarcado de monitoramento e para isso precisa escolher um sistema operacional adequado às suas necessidades. O diretor da empresa tem sugestões pela escolha de um sistema operacional de uma grande empresa para o projeto do novo equipamento de segurança, todavia, ainda existem diversas dúvidas sobre qual seria a melhor opção. Nós do departamento de pesquisa e desenvolvimento (P&D) fomos chamados para balizar as opções e definir a melhor decisão. A tarefa é montar uma apresentação do tipo *pitch* (apresentação de no máximo cinco minutos com os elementos-chave, que tem como objetivo vender uma ideia ou produto) que apresente a proposta do sistema operacional embarcado a ser utilizado. Nesse cenário, existe uma grande incerteza sobre qual será o sistema operacional a ser utilizado, pois a escolha desse sistema pode ser considerada como padrão de projeto para a empresa. Portanto, trata-se de uma ação decisiva para o futuro da empresa e de nossa equipe.

O cenário inicial consiste em apresentar a escolha do sistema operacional definido pela equipe de P&D para os diretores da empresa. Nesta apresentação, deve-se ter foco nos principais ganhos da nossa escolha e ainda a descrição do protótipo que foi feito como prova de conceito. Caso queira fazer o banner, deve-se apresentar de forma gráfica os conceitos, sem recorrer a grandes blocos de texto. No formato banner ou *pitch*, a apresentação deve conter:

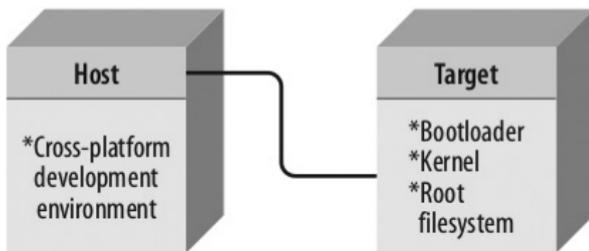
1. Descrição geral do sistema.
2. Vantagens e desvantagens da nossa opção de sistema operacional.
3. Custos.

A produção de um sistema embarcado necessita de persistência, pois as dificuldades de depuração e remoção de problemas de código são maiores que um sistema baseado em computador de uso geral. A compilação dos códigos não é feita diretamente no sistema embarcado e sim em um computador de uso geral e depois copiado para o sistema embarcado. Todo esse processo necessita de configurações e testes, dessa forma, a persistência e a criação de uma base de conhecimento são essenciais para que seja possível desenvolver produtos embarcados e visionar a internet das coisas.

Bons estudos!

Não pode faltar

O uso do computador é algo que está ligado ao cotidiano de todos, em todas as atividades o computador apresenta um papel importante. Todavia, essa presença tem vários modelos, tais como computadores pessoais, servidores, celulares e os sistemas embarcados. O hardware para sistemas embarcados possui características que diferem de equipamentos de uso geral (OLIVEIRA; ANDRADE, 2010). Em comparação a computadores de uso geral, podemos ressaltar as seguintes características:

- Limitação de memória principal e de armazenamento.
- Menor processamento.
- Interfaces de comunicação de mais baixo nível (como entradas e saídas analógicas e digitais).

Com isso, é de se esperar que os sistemas operacionais (SOs) para sistemas embarcados sejam diferentes em comparação aos computadores de uso geral. Dessa forma, Ganssle e Barr (2003) definem que sistemas operacionais embarcados são porções de *firmware* (software que é gravado diretamente na memória do sistema embarcado) que permitem multitarefas, ao comparar com a definição de sistema operacional do Tanenbaum (2008), que apresenta o sistema

operacional como uma camada de *firmware* que controla todos os dispositivos e apresenta uma interface menos complexa ao hardware.

Além das questões relacionadas ao hardware, temos que considerar que o processo de desenvolvimento de um sistema embarcado não é feito só e diretamente no sistema embarcado (sistema-alvo ou *target system*), existe um conjunto de ferramentas que são necessárias para que o *firmware* seja feito em uma arquitetura (*host system*) e depois aplicado no *target system*. Outra característica relevante é o processo de depuração (também conhecido como *debug*) que consiste na execução assistida do *firmware*, devendo ser embasada nas ferramentas que o sistema operacional embarcado possui, propiciando essa importante etapa do projeto.

Os sistemas operacionais embarcados também atuam como uma camada para facilitar o uso do hardware, todavia, o objetivo principal é ter um *firmware* que possa supervisionar e dividir o uso do hardware. Yaghmour (2003) apresenta os SOs embarcados que são alinhados ao hardware que será utilizado, levando em consideração o sistema-alvo (*target system*). A Figura 2.1 apresenta uma visão clássica do desenvolvimento de sistema embarcado, em que temos o *host* que possui todas as ferramentas de desenvolvimento e o *target* que possui o *bootloader* (responsável por receber novos códigos e inicializar os sistemas), o *kernel* que é parte do SO que conversa diretamente com o hardware e, por fim, o root *filesystem* que guarda os dados e executáveis do sistema.

Figura 2.1 | Forma de desenvolvimento de um sistema embarcado com SO

Fonte: Yaghmour (2003, p. 40).

Dessa forma, observando todos os aspectos de limitação de hardware e uso específico, os sistemas operacionais embarcados são elementos que possuem um conjunto de características gerais:

- Tamanho definido pelas limitações do hardware.
- Sistema de escalonamento de tarefas.
- Acesso básico ao sistema de entrada e saída para cada tarefa em execução.
- Utilização de bibliotecas e recursos.
- Estabilidade.
- Custo.
- *Toolchain* (componentes para geração e utilização do código).
- Prover ferramentas de debug.

Vale lembrar que a escolha do SO embarcado é uma das decisões mais importantes no projeto de um sistema embarcado. Diferente do computador de uso geral, em que as atualizações podem ser feitas de maneira "cotidiana", a atualização de sistemas embarcados, em diversas vezes, necessita de equipamentos específicos e uma pessoa treinada. Essa operação, se feita de maneira errada, pode tornar o equipamento inutilizável (em termos populares, pode tornar o equipamento *bricked*, que vem do inglês, tornar em tijolo, sem função). Dessa forma, além das características gerais, deve-se observar características finas do SO embarcado para a tomada de decisão (YAGHMOUR, 2013).

Algumas características mais específicas devem ser colocadas também para caracterizar os SOs embarcados, tais como as restrições ou aplicações de tempo. Um ponto relevante é como o SO deve tratar a ocorrência de algum evento. O SO embarcado não pode esperar que seja terminada alguma outra operação de entrada e saída ou algum outro evento.

Exemplificando

Considere um sistema que monitora e controla os sensores e os atuadores de uma prensa hidráulica que faz capôs de carro. O operador dessa máquina não pode ficar em certas áreas da máquina e não pode acionar essa prensa sem ter feito todos os procedimentos. Com isso, colocou-se uma rede de sensores óticos e sensores de pressão para monitorar essas áreas perigosas da prensa. Todos esses sensores têm como objetivo monitorar se o operador não está em áreas perigosas. Utilizando os sensores óticos e os sensores de pressão, tem o objetivo de verificar se os componentes da prensa estão nos locais corretos para a operação normal. O sistema embarcado NÃO pode esperar alguma

outra operação terminar, seja do próprio SO ou de outro *firmware*, para que o sinal vindo desses sensores seja tratado e em caso de emergência imobilizar o equipamento, esse atraso pode gerar problemas sérios ou ainda causar a morte de alguém.

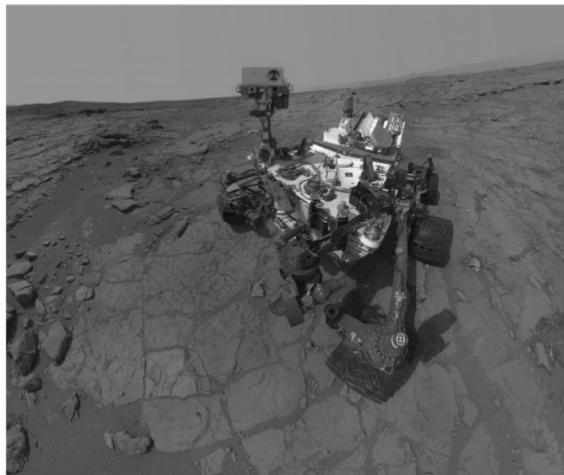
Dessa forma, esse SO deve ser do tipo *hard real time* (também conhecido por *immediate real time*). Com isso, qualquer ação será configurada para que quando ocorrer tenha precedência e seja tratada pelo SO e, por sua vez, pelo *firmware*. Existem os sistemas *soft real time*, em que é possível tolerar certos atrasos. A escolha das restrições de tempo que devem ser aplicadas está relacionada diretamente ao projeto, sendo que tal opção pode definir se o produto final terá sucesso ou não.

Os sistemas embarcados têm por definição o acesso a comunicações dos mais diversos tipos, pode-se fazer a comunicação entre sensores de temperatura, motores e outros elementos. Com isso, é possível tomar decisões com base nesses dados, essas decisões podem ou não suportar certo atraso na sua execução. Um sistema monitora a temperatura de uma sala de reuniões, quando a temperatura chega a um certo nível, as persianas são fechadas automaticamente para evitar a entrada do sol, diminuindo custos com o sistema de ar-condicionado. Nesse sistema, um atraso de poucos segundos não terá influência severa no resultado final e não existem vidas em risco. Como exemplo, o projeto que pode ser consultado no link a seguir. Disponível em: <http://repositorio.uniceub.br/bitstream/235/8693/1/21114229.pdf>. Acesso em: 30 nov. 2017.

Quando falamos de sistemas embarcados com SO, chegamos a verificar que é possível desenvolver diversos dispositivos, e nos tempos atuais, é muito fácil com acesso à Internet obter a informação de como construir esses produtos. Todavia, é necessário ficar atento a um elemento sério na construção desses elementos, a precisão da tarefa a ser executada. Como em sistemas embarcados temos a possibilidade de utilizar atuadores, sensores, entre outros, é necessário ficar claro que um *bug* no *firmware*

pode causar danos físicos a pessoas e propriedades. Nesse papel, nenhum dos processos de desenvolvimento, tanto hardware ou software, deve ser negligenciado. Com isso, quais seriam as estratégias para integrar esses dois elementos e quais metodologias aplicar?

Como explicado por Ganssle (1999), essas escolhas de SO também devem ser consideradas para medir e definir a produtividade do *firmware*, e uma frase de grande impacto citada por Ganssle (1999, p. 14.) é que "*firmware* é uma das coisas mais caras do universo", frase um tanto forte, todavia, com a capacidade de inserir os códigos no hardware gerou grande importância. Entretanto, o tamanho em bytes do SO e dos programas que serão executados tem grande importância.

Outro elemento que deve ser considerado é a capacidade e a necessidade do projeto com relação à rede de comunicação. Certos tipos de interfaces de comunicação apresentam necessidades específicas para o sistema operacional, redes do padrão wi-fi, *bluetooth* e outros padrões trazem grande complexidade ao *firmware*, caso o SO não possa controlar os elementos básicos dessas interfaces. Dessa forma, essa escolha está ligada ao suporte do SO e também como é feito o suporte da empresa que produz essa interface para o SO.

O SO possui a tarefa de prover acesso ao hardware sem que seja necessário que a aplicação tenha esse conhecimento específico. Sistemas de armazenamento, interface de rede e depuração são exemplos de elementos que o SO provê.

Um elemento final para o SO é como ele consegue apresentar elementos básicos de segurança ao hardware, com relação ao controle de temperatura, *watch dogs* (monitoramento do hardware, sob e subtensão etc.). Esses elementos possuem custo de processamento, mas podem evitar problemas e ainda facilitar o desenvolvimento do *firmware*.

A utilização de sistemas operacionais embarcados é fundamental para a produção de novos produtos, não sendo necessário se preocupar com todos os elementos do desenvolvimento. Celulares e

tablet são dispositivos que possuem um sistema embarcado, esses dois exemplos são os dispositivos mais gerais que se pode pensar. Todavia, existem outros elementos que possuem sistemas operacionais e são sistemas embarcados, como o robô da NASA Curiosity. Ele é um sistema embarcado que utiliza um SO chamado *Wind River VxWorks*, esse sistema é em tempo real e possibilita todo o controle. A Figura 2.2 apresenta uma imagem desse robô.

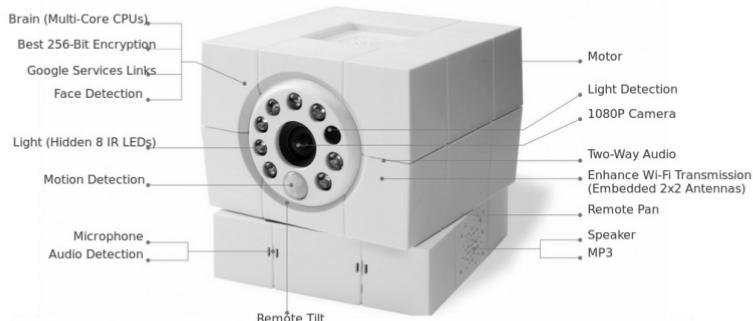



Figura 2.2 | Robô Curiosity da NASA, que possui um SO embarcado

Fonte: <https://mars.nasa.gov/msl/news/pdfs/MSL_Fact_Sheet.pdf>. Acesso em: 12 out. 2017.

Um caso mais comum são os equipamentos comumente conhecidos dos roteadores wi-fi. Esses equipamentos têm por capacidade receber uma conexão vinda normalmente de um provedor de acesso à internet e com interfaces wi-fi, criando um ponto de acesso. Um caso de grande interesse é o projeto *OpenWrt*, disponível em: <https://openwrt.org/>. Esse projeto disponibiliza diversos *firmwares* que podem ser instalados em roteadores comerciais. Outro exemplo é a câmera *Amaryllo*, que possui um SO GNU/Linux embarcado, em que é possível, pela vasta quantidade de recursos, prover acesso à câmera via Skype e outros elementos. A Figura 2.3 apresenta a imagem dessa câmera com sistema operacional embarcado.

Figura 2.3 | Câmera com monitoramento e diversos outros elementos de sistema embarcado

Fonte: <http://www.amaryllo.eu/n60104/prod_icare/>. Acesso em: 12 out. 2017.

Todas essas preocupações com relação ao tamanho do sistema operacional e ao *firmware* inserido no sistema embarcado nos tempos atuais, ainda é válida, ou seja, hoje temos diversos sistemas embarcados que possuem alguns MB de memória disponível e, mesmo assim, devemos pensar em códigos pequenos e SO com recursos mínimos. Hoje, temos hardwares de 5 cm por 5 cm que conseguem rodar o Microsoft Windows 10. Necessita-se verificar como poderíamos tratar certos problemas/ projetos e quais seriam aplicáveis nesses computadores de uso geral ou em sistemas embarcados.

Existe uma sequência de passos para a produção de um sistema embarcado com sistema operacional. O primeiro passo consiste em configurar as ferramentas de desenvolvimento em SO do *host*. De forma geral, as opções GNU/Linux têm mais opções nessa área (YAGHMOUR, 2003). De acordo com Phung (2008), um conjunto básico de software deve conter:

1. Bibliotecas do *kernel* do SO embarcado.
2. Sistema de arquivos para o sistema embarcado.
3. Compiladores para o sistema *target*.
4. Bibliotecas para uso no sistema embarcado.

Em mãos dessas ferramentas, o processo se inicia em desenvolver uma aplicação para ser executada no sistema embarcado e compilar essa aplicação usando o processo de *cross-compiling*.

O *cross-compiling* consiste em compilar um código no host para o *target*, onde o *host* e o *target* possuem arquiteturas diferentes (como o computador possuir arquitetura x86_64 e o sistema *target* é um Advanced RISC Machine ARM) (GANSSLE; BARR, 2003). Feita a compilação do *firmware*, é necessário configurar o sistema operacional para ser executado no hardware do *target*, e uma opção é o *busybox*. O processo consiste na compilação e na configuração básica de sistema de *bootloader* e sistema de arquivos e, por fim, o envio desse código para a plataforma *target*. Esse processo de envio depende de cada plataforma de desenvolvimento.

Pesquise mais

O vídeo do link a seguir mostra os conceitos básicos do *OpenWrt*, apresentando o seu sistema embarcado e quais são as suas funcionalidades. Disponível em: <https://www.youtube.com/watch?v=yfg7LsirCVg>. Acesso em: 21 nov. 2017.

O texto do link a seguir apresenta um resumo das características do robô Curiosity, da NASA, que também é um sistema embarcado com SO. Disponível em: <https://www.windriver.com/announces/curiosity/Wind-River_NASA_0812.pdf>. Acesso em: 21 nov. 2017.

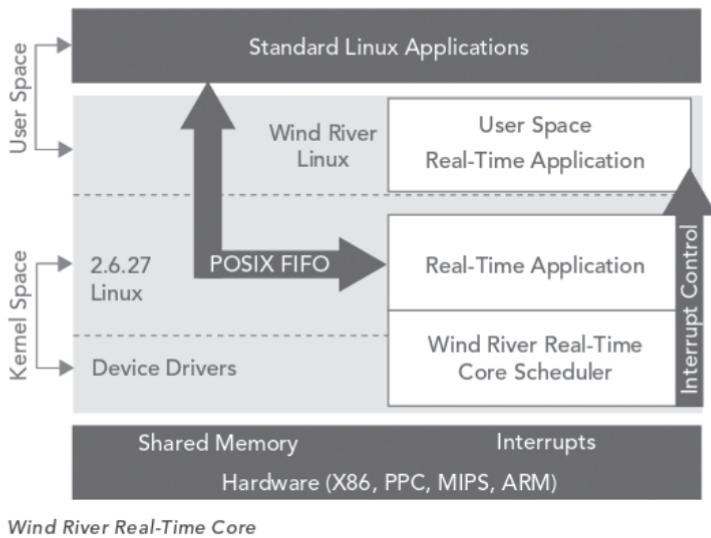
Sem medo de errar

A EmbarconEDU tem como objetivo desenvolver um novo projeto de sistema embarcado para monitoramento por sensores infravermelhos, câmeras e atuadores. O problema em questão consiste na escolha do sistema operacional, que deve levar em conta os seguintes requisitos:

1. Sistema que cuida da segurança de um ambiente. Dessa forma, deve ser robusto e ter respostas imediatas.

2. O custo do projeto pelas escolhas de software deve ser considerado.

3. A escolha impactará diretamente no sucesso do projeto, pois isso leva em conta todo o desenvolvimento.

4. O tempo de desenvolvimento é uma das preocupações.

Nesse caso, o que se deve considerar em primeiro momento é o tipo de SO, sendo que nesse cenário, podemos utilizar um *hard real time*, pois o monitoramento de sensores e afins deve ter prioridade. O sistema de escalonamento e estabilidade deve ser levado em conta, pois sendo um sistema de segurança, deve ter a maior disponibilidade possível.

O SO escolhido deve conter todas as ferramentas e as bibliotecas de desenvolvimento para os equipamentos de monitoramento. Essa escolha é importante, pois a escolha errada nesse requisito pode gerar trabalho e, além disso, a possibilidade de incompatibilidades diminui. As opções de código aberto e gratuitas devem ser escolhidas caso existam pessoas na equipe que já possuem experiência nesse cenário, pois em caso de necessidade de suporte em geral, não existe canal formal e com respostas rápidas. Uma opção é utilizar um sistema baseado em GNU/Linux, como o *Wind River*, pois é uma solução de baixo custo, em que as configurações corretas garantem a execução das tarefas no tempo necessário.

O *Wind River* possui diversas características que atendem ao projeto, tais como:

1. Suporte a controle de prioridades e controle de interrupções dedicados para sistemas de tempo real. A Figura 2.4 apresenta a camada de aplicações do sistema operacional (*Standard Linux Applications*), que se conecta à pilha clássica na camada de *User Space*, que, por sua vez, conecta-se à parte de real time e ao escalonador e, por fim, no hardware. Para o tratamento de eventos de tempo real, existem ligações diretas para que a aplicação consiga acesso mais rápido ao hardware.

2. Bibliotecas e sistema para desenvolvimento disponibilizados pelo fabricante propiciam menor tempo de desenvolvimento, no caso do *Wind River*, é disponibilizado pilhas e cadeias completas de programação.

3. Com *Wind River* não existe custo com o sistema operacional e, caso necessário, é possível comprar o suporte.

4. A documentação para a versão com ou sem suporte é a mesma.

Figura 2.4 | Parte central do sistema de tempo real do *Wind River*

Fonte: adaptada de Wind River (2009 p. 2).

Avançando na prática

Sistema para controle de lavanderias

Descrição da situação-problema

Uma empresa de lavanderias inovadora, chamada 128ASEC, quer automatizar todo o processo de limpeza das roupas. A aplicação de um sistema embarcado com sistema operacional foi levantada e deve controlar sensores de nível de água, pressão e atuadores, como válvulas solenoides e bombas de água. Todos esses sistemas serão responsáveis por controlar toda uma lavanderia. Deve-se levar em conta que se o sistema falhar, podem ocorrer inundações e sobrecargas em equipamentos, levando riscos ao patrimônio e às pessoas. Uma empresa com pouca experiência está sugerindo que sejam utilizados sistemas baseados em computadores do tipo *personal computer* clássicos, com sistemas operacionais do cotidiano. O seu papel é apresentar em cinco slides os diferenciais e trazer o projeto para nossa empresa.

Resolução da situação-problema

Você deve apresentar os conceitos básicos de sistemas operacionais embarcados, tais como:

- Tamanho definido pelas limitações do hardware, isso delimita qual SO escolher.
- Acesso básico ao sistema de entrada e saída para cada tarefa em execução.
- Utilização de bibliotecas e recursos, esse item define quanto tempo será gasto na base do projeto.
- Estabilidade, por ser um hardware e SO dedicados, minimiza as falhas e as atualizações constantes em um projeto baseado em PC.
- Custo, em relação ao SO, pode ser em sistema aberto e gratuito, visando que a equipe possui experiência. Um SO proprietário pode ser mais indicado com relação à experiência e ao suporte oficial.
- Uso dedicado, o uso dedicado é um dos pilares da estabilidade do sistema, sendo que teremos certeza que não terá outras tarefas a tratar.

Dada a explicação geral, deve-se pontuar os aspectos de controle de hardware em casos de falhas, as formas de acesso direto ao hardware e ainda o aspecto simplista de um sistema operacional embarcado, que minimiza diversas possibilidades de *bug* e falhas de segurança que podem ocorrer com sistemas operacionais de uso geral. Em explicações mais simples, um sistema embarcado com SO dedicado para sistemas embarcados não reiniciará para aplicar uma atualização e ainda provê os mecanismos de controle dos equipamentos.

Faça valer a pena

1. Os sistemas embarcados são elementos que fazem parte do cotidiano das pessoas, mesmo que não seja notado por todos. Esses sistemas têm importância na vida de todos e em alguns casos podem ser responsáveis por salvar vidas. Tais sistemas embarcados, em diversos casos, podem ter sistemas operacionais que têm diversas tarefas.

Dentre as afirmações a seguir, assinale aquela que é uma função do sistema operacional embarcado:

a) Controlar as trocas de contexto para a execução de processos.

b) Fornecer elementos e interfaces para a depuração do firmware.

c) Apresentar os elementos críticos de memória de um firmware.

d) Fornecer acesso ao sistema de arquivos.

e) Prover um bootloader para garantir o acesso ao hardware.

2. A proteção de sistemas mecânicos e elétricos em cenários críticos é feita por sistemas embarcados autônomos. Esses equipamentos de proteção têm papel de grande importância, utilizando sensores e atuadores para verificar e monitorar elementos de controle e de segurança, em caso de falhas, o sistema embarcado deve ser capaz de imobilizar o equipamento.

Dados os cenários de proteção de equipamentos, avalie as seguintes afirmações:

I – Os sistemas operacionais devem ser de tempo real hard.

II – O sistema operacional deve priorizar os sistemas de entrada e saída.

III – Os sistemas de tempo real soft são os mais indicados nesse cenário.

IV – O sistema operacional embarcado deve priorizar todos os elementos de *firmware* para monitorar os sensores e os atuadores.

É correto o que se afirma em:

a) As afirmações II, III e IV estão corretas.

b) As afirmações III e IV estão corretas.

c) Apenas a afirmação I está correta.

d) Apenas a afirmação II está correta.

e) As afirmações I e IV estão corretas.

3. Ao se programar um equipamento embarcado com sistema operacional, temos um cenário que programamos o *firmware* em uma arquitetura e enviamos esse código compilado para o hardware do sistema embarcado. Todo esse processo envolve alguns elementos que o sistema operacional para o sistema embarcado deve fornecer para se tornar uma ferramenta que aumenta a produtividade.

Dado o cenário, aponte qual das alternativas apresenta os elementos principais de um sistema operacional embarcado:

a) *Kernel*, ferramentas básicas do SO, *bootloader* e root *filesystem*.

b) Atuadores e sensores.

c) *Kernel, bootloader* e *root filesystem*.

d) *Host* e *target*.

e) *Target* e *bootloader*.

Seção 2.2

Sistemas de arquivos embarcados

Diálogo aberto

O grupo de pesquisa chamado *LightCatch* é liderado pelo gestor de inovações que está trabalhando em um projeto de pesquisa para produzir um sistema de detecção de feixe de partículas. Esse sistema é baseado em um canhão de elétrons e um sensor, ambos montados em uma caixa de aço. Ligados ao sensor existe um conjunto de fios que são conectados a um sistema embarcado com sistema operacional, que possui diversas entradas do tipo analógico-digital (AD) (entrada que recebe um sinal analógico que consiste em variação de tensão e converte para um valor decimal, neste caso, entre 0 até 65536). Quando diversos elétrons colidem no sensor, transformam-se em diversos sinais e esses sinais são enviados às entradas AD do sistema embarcado, ao receber esses sinais, é necessário processá-los e transmiti-los pela interface de rede da placa de SO embarcado. O processamento envolve cálculos estatísticos básicos (média, desvio padrão e moda) e formatação dos dados em *Flexible Image Transport System* (FITS, formato de dados para imagens e dados científicos).

Você faz parte desse grupo de pesquisa e está responsável por construir o sistema embarcado para receber os sinais, processar os dados e enviar pela interface de rede. O gestor de inovações insiste em produzir um sistema embarcado baseado em um microcontrolador, sem sistema operacional para fazer esse detector. O seu trabalho é preparar uma apresentação de cinco minutos para convencer o gestor de inovações e seus colegas sobre o uso de SO nesse projeto embarcado. A apresentação deve conter:

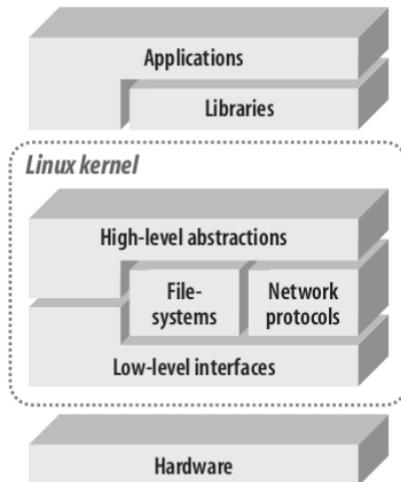
1. As vantagens do sistema embarcado com sistema operacional.

2. Quais componentes de um SO embarcado representam vantagens em relação a um sistema embarcado sem SO.

3. Quais sistemas de arquivos são indicados para esses cenários.

Não pode faltar

Os sistemas operacionais (SOs) embarcados têm diferenças pontuais comparados aos sistemas operacionais de uso geral. Todos esses pontos convergem para que o SO embarcado possua um tamanho muito menor que os sistemas de uso geral, como o *BusyBox*, que é baseado em um sistema Linux e tem de 1 MB a 5 MB (ZHANGJIN; ZIQIANG, 2013). Esse espaço reduzido é relacionado à limitação do hardware do sistema embarcado e não pode ser deixado de lado em nenhum momento, essa limitação que é a essência do sistema embarcado.

Reflita

Existem diversos sistemas operacionais embarcados, cada um com conceitos e objetivos específicos. Os sistemas *Unix-like* possuem características bem definidas com relação ao seu tamanho reduzido, porém, as ferramentas e os processos de desenvolvimento são menos dinâmicos e mais manuais. Os sistemas da Microsoft para *Internet of Things* (IoT) possuem um tamanho maior comparado aos sistemas *Unix-like*, porém seu desenvolvimento é gerenciado por *Integrated Development Environment* (IDE). De forma geral, cada solução possui sua aplicabilidade. Assim, é razoável definir apenas um tipo de solução para todos os problemas ou uma visão mais abrangente é mais indicada?

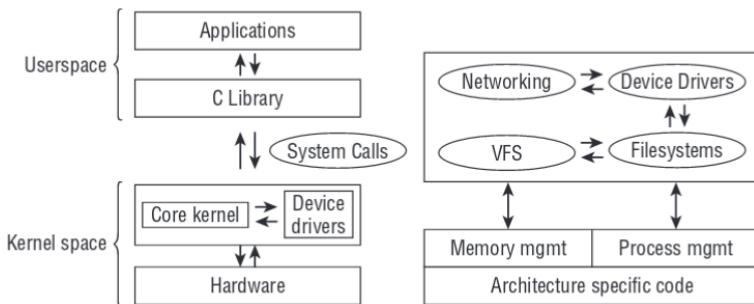
A Figura 2.5 apresenta uma visão geral dos componentes de um SO relacionado ao sistema *UNIX* (família de sistemas operacionais que descreveu os padrões onde diversos SOs se baseiam, tais como os sistemas GNU/Linux). O primeiro nível apresenta as aplicações, que acopladas às bibliotecas de sistema, podem ser executadas pelo sistema operacional, o *kernel*, que representa a interface que conversa com o hardware e possui duas camadas de abstração, uma para as aplicações e outra para o hardware, ambas utilizam os sistemas de arquivo e protocolos de rede e, por fim, o hardware que utiliza todas as camadas. Os SOs embarcados diferem de aplicações de computadores de uso geral, as bibliotecas são menores e o suporte a sistemas de arquivos e bibliotecas é limitado (BELL, 2016). Como exemplos em aplicações GNU/Linux, utiliza-se como biblioteca padrão em programas em C a *glibc*, e em sistemas embarcados, a *uClibc*, versão da *glibc*, que possui cerca de 2 MB (YAGHMOUR, 2003). Com a aplicação pronta, utiliza-

se o *kernel* para fazer acesso ao hardware, como o sistema é dividido em camadas, todos os elementos ficam encapsulados para que o acesso possa ser feito de forma padronizada, sem que seja necessário se adentrar nos detalhes de diferentes fabricantes de cada hardware (YAGHMOUR, 2003).

Figura 2.5 | Visão geral de sistema com GNU/Linux

Fonte: adaptada de Yaghmour (2003, p. 44).

Um sistema Microsoft Windows 10, em sua instalação, pede 20 GB de espaço em disco para parte básica (BORYCKI, 2017), o *BusyBox* ocupa cerca de 5MB para prover acesso básico ao hardware, por sua vez, o Windows IoT necessita de no mínimo 2 GB.

As aplicações com um SO embarcado não precisam ter acesso direto a diretrizes de hardware, sendo que o SO provê todo um conjunto de elementos, e um dos elementos básicos são os itens de um sistema de arquivos de um sistema embarcado. Como exemplo, em sistemas *Unix-like*, o sistema de arquivo contém um conjunto de diretórios que provê acesso aos arquivos e também acesso a diversos elementos de hardware.

Pode-se dividir um *kernel* de sistema operacional em quatro partes (MAUERER, 2010):

1. Controle de IO (redes, USB etc.).
2. Controle de armazenamento.
3. Gerenciamento de processos.
4. Gerenciamento de memória.

A parte de controle de processos e controle de memória são os pilares do *kernel*, eles são os responsáveis por garantir o acesso segmentado, protegido e hierarquizado da memória do sistema (sendo memória RAM, entre outros) e ainda o escalonamento de processos, sendo responsável por quais tarefas serão executadas a cada momento (TANENBAUM, 2008). A parte de redes conversa com o controle de dispositivos, sendo ela responsável por utilizar os *drivers* (elementos de software que são acoplados ao *kernel* para prover o acesso personalizado a um certo hardware) e fazer as chamadas de sistema e, com isso, utilizar os componentes de hardware para a comunicação (YAGHMOUR, 2003). A Figura 2.6 apresenta uma visão mais precisa de um sistema operacional, em que o espaço de usuário consiste no software do usuário e bibliotecas do sistema. Com as chamadas de sistema, é possível acessar as camadas do *kernel* e com o auxílio dos *drivers*, é possível acessar o hardware, em paralelo ao subsistema de rede. O sistema de arquivos conversa com o sistema de gerenciamento de memória e processos que, por sua vez, utilizam porções específicas de cada arquitetura para acesso ao hardware.

Quando se fala de sistemas embarcados, deve-se levar em conta que cada um desses módulos possui elementos dedicados ao hardware alvo, por exemplo, a parte de rede só consegue acesso aos dispositivos presentes ou planejados, sendo que não existem *drivers* para outros dispositivos, a parte de *Virtual File System* (VFS) e *File system* também é dedicada, módulos e *drivers* para sistemas de arquivos não planejados não estão presentes (YAGHMOUR, 2013). A visão detalhada do sistema operacional em sistemas embarcados é necessária para manter apenas as partes que serão utilizadas e utilizar espaço para manter recursos que nunca serão demandados. Com essa visão, é possível encontrar lugares onde se pode remover o código e outros elementos do SO e liberar espaço para a aplicação. Com isso, deve-se literalmente construir os SOs para o sistema *target*.

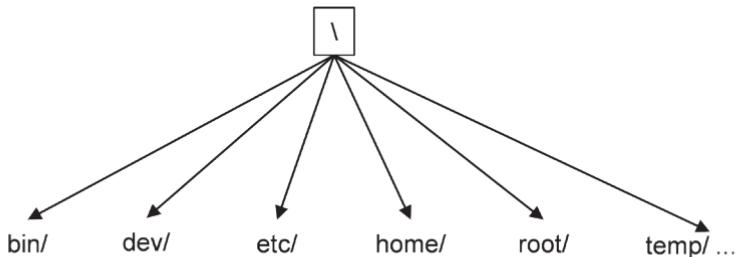
Figura 2.6 | Visão esquematizada de um sistema operacional

Fonte: adaptada de Mauerer (2010, p. 4).

Os sistemas operacionais embarcados possuem os mesmos elementos que um sistema operacional de uso geral. No entanto, alguns elementos são sumarizados, tais como o conjunto de *drivers* para dispositivos anexos e interfaces, acesso a diferentes sistemas de arquivos e bibliotecas de sistema e de dispositivos.

Dados esses elementos principais do *kernel*, para que seja possível utilizar o SO, deve existir um conjunto básico de diretórios que contém as bibliotecas, o comando e outros elementos. Essas ferramentas representam uma grande vantagem a um sistema embarcado sem SO, pois é possível inspecionar de forma mais precisa o software que está executando. Como exemplo de um sistema baseado em GNU/Linux, teremos (GANSSLE; BARR, 2003):

- bin: arquivos binários básicos para o sistema.
- boot: *bootloader* do sistema.
- dev: arquivos que definem cada dispositivo do sistema.
- etc: arquivos de configuração.
- home: arquivos dos usuários.
- lib: diretório que contém as bibliotecas do sistema.
- media: contém os dados mapeados de sistema de armazenamento.
- opt: localização de softwares adicionados como opcionais.
- proc: informação sobre o *kernel*.
- root: dados do administrador do sistema.

- sbin: binários essenciais ao sistema.
- sys: sistema virtual com arquivo de controle.
- tmp: arquivo de sistema temporário.
- usr: documentação e outras aplicações do usuário.
- var: logs e informações do serviço.

Exemplificando

O *BusyBox* é um conjunto de ferramentas para o desenvolvimento de sistemas embarcados, ele possui todos os diretórios e as estruturas que um sistema GNU/Linux para computadores de uso geral possui. Todavia, tudo no *BusyBox* é empacotado, implementado ou compilado, de forma a ocupar o menor espaço possível, gerando imagens pequenas, como de 4 MB a 8 MB. Sistemas de arquivo, *drivers* e até ferramentas de sistemas são mais simples para ocupar menos espaço.

A estrutura de um sistema embarcado baseado em GNU/Linux é formado como uma árvore, onde existe um diretório, que é o início de todos os diretórios que são derivados dele. A Figura 2.7 mostra um exemplo de como são estruturados os diretórios e os arquivos. Cada diretório possui elementos de importância, por exemplo, o boot possui papel principal para o sistema embarcado, apresentando os arquivos de *bootloader* (elemento que tem um binário compilado para o sistema-alvo, que consegue fazer a inicialização do sistema) e o *kernel* compilado, que possui todos os códigos necessários para utilizar o hardware (YAGHMOUR, 2003). No diretório/boot pode existir mais de uma imagem do *kernel*.

Figura 2.7 | Visão geral do sistema de arquivos de um sistema embarcado

Fonte: elaborada pelo autor.

Nos sistemas da Microsoft para IoT, a estrutura é diferente. Essa diferença vem da própria filosofia de desenvolvimento. No caso do Windows, utiliza-se uma parte da IDE Visual Studio para desenvolver os equipamentos. Os dados são armazenados em diferentes partições, onde:

- C:\: Todos os dados de sistema ficam armazenados, incluindo bibliotecas.
- Data\: Arquivos dos usuários.
- EFISP: *Bootloader* e configuração de inicialização (BORYCKI, 2017).

As bibliotecas para sistemas embarcados para Microsoft Windows IoT são baseadas no *.NET Framework*, que representa um grande conjunto de bibliotecas que proporciona todo o acesso necessário ao hardware (BORYCKI, 2017).

Para aprofundar-se no tema de sistemas embarcados com sistema operacional, assista ao vídeo a seguir. Disponível em: <https://www.youtube.com/watch?v=Ie3ICecqG3g>. Acesso em: 17 nov. 2017.

A Visual Studio para IoT é disponível para download e existem diversos tutoriais para a produção de um sistema embarcado. Disponível em:

<https://developer.microsoft.com/en-us/windows/iot>. Acesso em: 17 nov. 2017.

No processo de desenvolvimento de um sistema embarcado com sistema operacional, em primeiro lugar, deve-se escolher a base do seu produto, com isso, preparar o ambiente e escolher quais serão os componentes que serão utilizados. Na escolha dos componentes, deve-se levar em conta os recursos de hardware disponível em seu sistema-alvo e as necessidades na execução do software. Dessa forma, escolher quais módulos, *drivers* e sistema de arquivos estarão disponíveis é um dos primeiros passos para desenvolver um sistema embarcado. Após essa etapa de seleção, enviar os arquivos para o *target* e executar os testes necessários para verificar se os recursos disponibilizados pelo SO atendem às necessidades do software e dos usuários.

Sem medo de errar

O líder de inovação da empresa *LightCatch* tem um projeto para a construção de um sistema para detecção de feixes de elétrons e quer um sistema apenas com um microcontrolador. O seu objetivo é apresentar argumentos para a utilização de um sistema embarcado com sistema operacional, sendo as características principais:

1. Captura de diversos sinais analógicos.
2. Processamento de dados.
3. Formatação dos dados.
4. Envio pela rede desses dados processados e formatados.

A primeira colocação necessária consiste em apresentar as vantagens de um sistema embarcado com sistema operacional. Sobre essas vantagens, dado o projeto, podem-se destacar:

- Capacidade de troca de contexto do processo.
- Como existe um conjunto de *drivers* de acesso ao hardware no SO embarcado, a utilização de componentes de rede se torna menos complexa, pois o SO faz o acesso básico e a aplicação não precisa tratar desse tipo de processo.
- Como é necessário processar e formatar dados, o uso de bibliotecas de software se torna mais abrangente pela utilização do SO.
- Dado o uso de SO e os seus módulos, é possível utilizar as bibliotecas para fazer a depuração do código.

Pode-se contextualizar com sistema *Unix-like* como o *BusyBox*, apresentando características como:

- Tamanho reduzido.
- Sistema baseado em Linux, que tem por tradição sua estabilidade.
- Sistema maduro e com diversas placas e sistema embarcado homologados.

Em sistemas da Microsoft, pode-se comentar sobre elementos, tais como:

- Suporte oficial e documentação centralizada.
- Ferramentas de desenvolvimento integrada (Visual Studio).
- Diversas placas com homologação oficial da Microsoft.

Prepare sua apresentação em slides ou banner, para ajudar a guiá-lo durante a apresentação.

Avançando na prática

Sistema embarcado com sistema operacional para monitoramento de qualidade elétrica

Descrição da situação-problema

O data center do Banco BemInvestido está com problema em sua qualidade na energia elétrica, especificamente com interrupções elétricas constantes e curtas. Os sistemas de nobreak têm tido sucesso em evitar problemas nos computadores. O gerente de operações do data center quer produzir um relatório para enviar à empresa de distribuição elétrica com o objetivo de apresentar os problemas e demonstrar os custos que podem surgir se uma falha com tempo maior ocorrer. O gerente de operações do data center conversou com uma empresa para a produção de um dispositivo baseado em um sistema embarcado com sistema operacional, porém, existe uma preocupação grande sobre a compatibilidade do hardware básico do sistema com o sensor que fará a leitura da tensão elétrica. Dessa forma, é necessário produzir um relatório de uma página para explicar como resolver essa questão do acesso aos dispositivos. Esse relatório deve explicar os elementos básicos de um sistema operacional embarcado e apresentar o papel dos *drivers* e dos módulos do *kernel* para acesso dos dispositivos.

Resolução da situação-problema

Nesse relatório, deve-se apresentar os conceitos básicos de um SO para sistemas embarcados, tais como as camadas do *kernel*, as bibliotecas de sistema e os *drivers*/módulos. O papel dos módulos deve ser enfatizado, como as pontes que ligam o hardware específico com o *kernel*, tornando a aplicação mais simples e deixando a complexidade de acesso aos dispositivos para o *kernel* e descrever que o sistema operacional embarcado deve ser homologado para o hardware e ainda ter os *drivers* para esse sensor de tensão elétrica.

Faça valer a pena

1. Uma empresa está produzindo um sensor de chuva com sistema operacional para detectar quais áreas têm o melhor cenário para fazer a produção de maçãs. Esse sensor deve capturar os níveis de água e enviar os relatórios via uma rede WiMax (Worldwide Interoperability for Microwave Access, rede de wi-fi para longas distâncias).

Qual(is) módulo(s) de um sistema operacional embarcado possui(em) o objetivo de prover acesso básico ao dispositivo de rede?

a) *Kernel* e *drivers*.

b) Sistema de arquivos.

c) Interface gráfica.

d) *Kernel.*

e) *Drivers* de acesso ao *bootloader.*

2. Um freelancer fez um projeto para um sistema embarcado sem sistema operacional, que possui cerca de 500 linhas de código e cobrou cerca de duas vezes mais que um sistema operacional com 200 linhas de código, e ainda com tempo de desenvolvimento duas vezes menor que o produto sem sistema operacional.

I- Um sistema embarcado com sistema operacional ajuda a tratar a complexidade de acesso aos dispositivos.

II- Sistemas embarcados com sistema operacional ocupam espaço que poderia ser utilizado pela aplicação.

III- Os *drivers* para um sistema operacional embarcado não devem ser utilizados.

IV- O desenvolvimento de um sistema embarcado com SO e seus módulos provêm elementos de depuração.

É correto o que se afirma em:

a) As afirmações II, III e IV estão corretas.

b) As afirmações I, II e IV estão corretas.

c) As afirmações I e III estão corretas.

d) Apenas a afirmação I está correta.

e) Apenas a afirmação IV está correta.

3. Os sistemas embarcados possuem duas grandes vertentes, podendo ser baseadas em Unix-like ou sistemas Microsoft Windows IoT. Ambas as soluções têm características e aplicações diferentes, onde cada cenário possui diretrizes diversas.

Em um sistema embarcado baseado em Microsoft Windows IoT, a biblioteca principal é baseada em:

a) *.NET framework.*

b) *uClibc.*

c) *glibc.*

d) *OpenCV.*

e) *OpenGL.*

Seção 2.3

Dispositivos de armazenamentos embarcados

Diálogo aberto

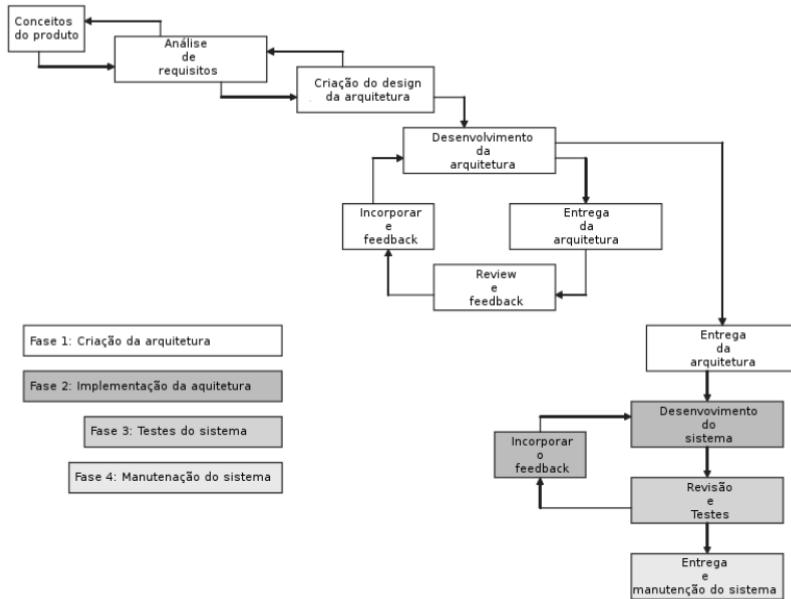
Na seção anterior, estudamos os elementos do *kernel* do sistema operacional embarcado, bem como suas interfaces para controle de dispositivos (módulos e *drivers*) e sua estrutura de dados onde são armazenados os arquivos de configuração, softwares e outros itens relacionados. O próximo passo é entender as formas para que seja possível conversar com o hardware e armazenar todos esses arquivos.

Nesta etapa, serão apresentados os mecanismos para utilizar os sistemas que controlam e estruturam a memória de um sistema embarcado. Os diferentes tipos e formas, sempre ressaltando a hierarquia de memória e suas relações de custo.

Há 12 anos, ocorreu uma grande chuva e os bueiros da rua não foram capazes de escoar a água. Com isso, seu carro foi inundado, ocasionando grandes danos. Atualmente, um engenheiro eletricista terminou sua faculdade e fundou uma empresa e seu primeiro projeto é um sistema que consiga monitorar o nível de água de bueiros e enviar avisos para os celulares das pessoas próximas sobre os possíveis problemas de inundação. Todavia, o problema agora está em definir o tipo de memória a ser utilizado no projeto, pois é necessário armazenar todos os eventos que ocorram. A primeira ideia seria utilizar um cartão do tipo *secure digital* (SD), circuitos com *flash translation layer* (FTL), em que é possível ter acesso de forma direta para o armazenamento. Entretanto, ao fazer as pesquisas, encontrou chips de memória *flash raw*, baseados em circuito *not-and* (NAND). O líder de pesquisa está com grandes dúvidas de qual tecnologia utilizar, seu papel é apresentar um quadro comparativo, contendo:

- comparação entre os dois tipos de memória: SD com FTL e NAND *raw flash*.

- implicações sobre o uso dessa memória com relação ao sistema operacional.

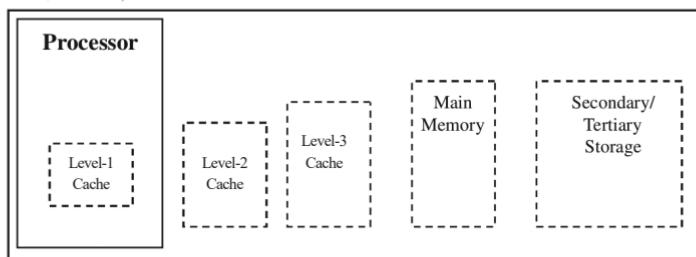
- tipos de sistema de arquivos para cada cenário.

Nesta etapa do estudo, serão apresentados os elementos necessários para que seja possível manipular os dispositivos de memória de um sistema embarcado, mostrando suas características e ainda considerando quais são os elementos dedicados para leitura e escrita desses dispositivos de armazenamento.

Não pode faltar

Todo projeto de sistema computacional é um processo que necessita de cuidado e atenção. Essa preocupação com sistemas embarcados é grande, pois existem elementos que não podem ser alterados depois de entregues ao cliente.

O processo de desenvolvimento de um sistema embarcado envolve diversas etapas e por ter um hardware que é enviado para o cliente e usos específicos (sem acesso à internet ou áreas remotas), as atualizações são inviáveis. Conforme a Figura 2.8, todos os passos do projeto são interativos e cada etapa é refeita até atingir o nível de qualidade necessário. Em primeiro, na fase 1, se avaliam os conceitos do produto. Com base nisso, é feita a análise de requisitos (necessidades que são avaliadas dados os conceitos), na terceira etapa, é feita a criação do design da arquitetura do projeto e com esse projeto é feito o desenvolvimento, os testes e os feedbacks desse processo. No final dessa etapa, é criada a arquitetura final (entrega da arquitetura) e, com isso, inicia-se o desenvolvimento do sistema (fase 2), testado e validado até a entrega final (fase 3 e 4).

Figura 2.8 | Processo de desenvolvimento de um sistema embarcado

Fonte: adaptada de Noergaard (2012, p. 8).

Nesse processo completo, a escolha de um componente errado pode gerar problemas, e um dos principais componentes que se deve ter atenção é a memória. A memória de um sistema embarcado deve ser tratada de forma específica, pois seu tamanho e tipo diferem do uso geral, como em *personal computers* (PC) ou servidores. Segundo Yaghmour (2003), a primeira diferença consiste no seu tamanho reduzido pela própria arquitetura e organização e o segundo ponto consiste em que dispositivos embarcados utilizam memória do tipo *flash*. Segundo Ganssle (2008), *flash* é um tipo de memória não volátil que pode ser escrita e apagada por softwares e não necessita ser retirada do circuito para essas operações. Mesmo com essas peculiaridades, o sistema embarcado também possui um sistema hierarquizado de memória (NOERGAARD, 2012).

Podemos dividir a memória de um sistema embarcado em diversas áreas e aplicações, desde as divisões da memória do processador até a memória de trabalho e armazenamento (OLIVEIRA; ANDRADE, 2010). Todavia, uma divisão primária seria a memória do tipo ROM (*read-only memory*), que possui o sistema básico da placa que receberá o SO

operacional e do tipo NVRAM (*non-volatile random access memory*), que recebe o SO instalado na placa. Esses diversos tipos de memória são aplicados em todos os projetos embarcados (CAPELLI, 2010).

A Figura 2.9 apresenta que é possível encontrar memória em cada parte de um sistema embarcado. Normalmente, as memórias do tipo cache estão junto ao processador e a *Main Memory* é um circuito separado, bem como o segundo e o terceiro nível de memória. As memórias do tipo ROM e NVRAM são classificadas como os últimos dois níveis.

Figura 2.9 | Hierarquia de memória de um sistema embarcado

Fonte: Noergaard (2012, p. 223).

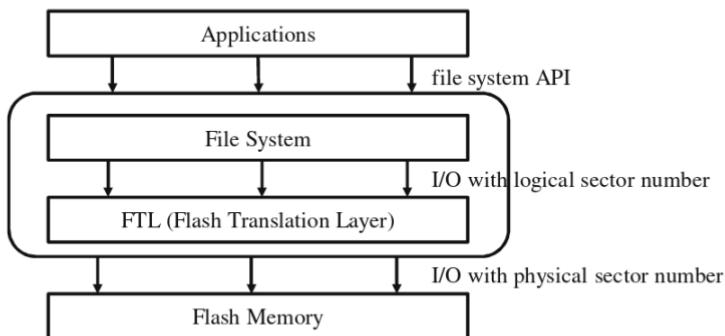
Um exemplo de hardware para sistemas embarcados é o *BeagleBone* Black, que é um equipamento de baixo consumo elétrico com 512 MB de RAM e com um processador ARM AM3358 da *Texas Instruments* com 32 KB de cache *Level 1*, 256 KB de cache *Level 2* e 64 KB de *Level 3*. Mais informações no link a seguir. Disponível em: <http://beagleboard.org/black>. Acesso em: 23 nov. 2017.

Os caches *Level* 1 e 2 estão ligados ao processador e são feitos com transistor do tipo metal oxide *semiconductor field-effect transistor* (MOSFET). Ambos são classificados como memória do tipo SRAM (*static random access memory*) (NOERGAARD, 2012). Os caches *Level* 3 são áreas de memória junto ao processador que podem ser compartilhadas entre vários núcleos.

Essa hierarquia de memória difere em sistemas embarcados com ou sem sistema operacional. Em sistemas sem SO, consideramos a memória disponível no microcontrolador e algum módulo

externo, já em sistema com SO temos uma memória principal e outra para armazenamento.

Em sistemas baseados em GNU/Linux, considera-se que todos os dispositivos (*memória flash, hard disk,* câmeras etc.) são arquivos. Dessa forma, podemos dividir tais elementos de duas formas:

- Dispositivos de caracteres: tais elementos são escritos e lidos como um fluxo de dados, em que cada operação para acesso ou envio da informação é feita por unidade, não sendo possível acessar uma posição em específico (não é permitido o acesso aleatório). Interfaces de redes ou câmeras são exemplos desses dispositivos.
- Dispositivos de bloco: esses dispositivos também são tratados como arquivos, porém, o acesso de leitura e escrita podem ser feitos de forma aleatória. Dessa forma, não é necessário consultar todo o volume para escrever ou ler uma informação. Exemplos dessas interfaces são as memórias de armazenamento, como *hard disk* ou do tipo *flash*, com *Flash Translation Layer* (FTL).

Para o armazenamento, temos vários tipos de memórias que podem ser utilizadas, tais como discos rígidos, memórias *flash* com ou sem FTL. A escolha do tipo de memória em um sistema embarcado está ligada diretamente ao projeto. As memórias baseadas em discos rígidos ou *flash* com FTL possuem interfaces que garantem o acesso ao dispositivo como um dispositivo de bloco. Essas interfaces são os sistemas de arquivos, que são constituídos por um conjunto de estruturas de dados e algoritmos que mapeiam a área física do sistema de blocos para prover aos softwares os lugares para guardar os arquivos.

No caso do GNU/Linux, temos o *second extended filesystem* (ext2) ou o *third extended filesystem* (ext3), que são sistemas de arquivos com *journal*. O *journal* é uma área onde se grava todas as alterações que serão executadas no sistema de arquivos, pois caso ocorra alguma falha (como queda de energia durante a escrita), será possível recuperar os dados.

As memórias do tipo flash são mais utilizadas, pois não possuem peças móveis, tais como motores ou discos e ainda, podendo utilizar as memórias sem essa camada FTL, em que o acesso fica por conta do SO, deixando o custo de hardware menor, não sendo necessário o uso de outro chip para a memória RAM.

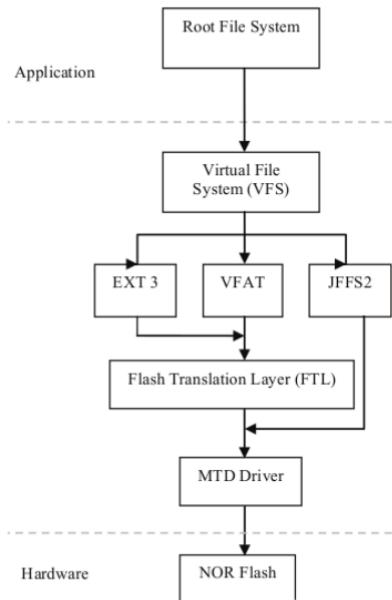
A Figura 2.10 apresenta uma visão geral da organização de um sistema de memória flash, em que as aplicações não necessitam ter conhecimento de todas essas camadas para a operação, apenas a informação de qual sistema de arquivos está em curso já é o suficiente. Quando a aplicação pede a escrita ou a leitura de um arquivo, o sistema de arquivos possui todas as informações lógicas das localizações desse arquivo no chip, porém, é a FTL que possui as informações em que os dados estão localizados fisicamente.

Figura 2.10 | Visão geral de um sistema de memória flash

Fonte: Chung et al. (2009, p. 1).

Todavia, existem cenários onde o custo, o tamanho físico e a confiabilidade de uma memória com FTL são proibitivos. Dessa forma, torna-se necessário o uso de memória flash do tipo *raw*, na qual a camada que pode prover as informações ao sistema de arquivos não existe, deixando essa tarefa ao sistema operacional.

Memória com FTL são todas aquelas que, além dos circuitos para armazenamento, ainda existe uma camada que já oferece ao sistema operacional acesso aos dispositivos, como SSD (Solid-state drive), MMC (MultiMediaCard), eMMC (embedded Multi-Media Controller), Reduced-Size MultiMediaCard (RS-MMC), Secure Digital (SD), mini SD, micro SD, USB flash drive, CompactFlash, Memory Stick, Memory Stick Micro.

Exemplificando

Uma memória da Micron Technology MT25QL02GC que pode armazenar 2 Gb tem 6 mm por 8 mm e possui o Tape Ball Grid Array (TGBA) como encapsulamento. Esse dispositivo pode armazenar as informações de um sistema embarcado, porém, o acesso a essa memória deve ser feito por algum sistema, como o SO, que pode gerenciar as localizações físicas das informações. Mais informações no link a seguir. Disponível em: <https://www.digikey.com/product-detail/en/micron-technology-inc/MT25QL02GCBB8E12-0SIT-TR/MT25QL02GCBB8E12-0SIT-TR-ND/6036307>. Acesso em: 23 nov. 2017.

Para que seja possível utilizar as memórias do flash do tipo *raw*, é necessário o apoio de algum tipo de software do SO para controlar as posições físicas da informação que não são providas pela FTL. Sem isso, seria muito custoso para a aplicação codificar o acesso a diferentes tipos de memória *raw* flash, onde a complexidade da aplicação subiria. No caso de sistemas GNU/Linux, é possível utilizar o *memory technology devices* (MTD). O MTD é uma parte do *kernel* do Linux, que provê acesso padronizado a diversos sistemas de memória *raw flash*.

O MTD provê todo um conjunto de software que garante o acesso a dispositivos do tipo *raw flash* de maneira padronizada, mostrando os arquivos em um sistema de arquivos dedicado a essa finalidade (DENG, 2017). Conforme a Figura 2.11, a camada de aplicação utiliza o root file system para armazenar os dados, as chamadas de escrita e leitura são passadas para o *virtual file system*, que provê acesso a diferentes sistemas de arquivos (EXT3, VFAT ou JFFS2), quando utilizado o EXT3 ou VFAT, o hardware possui a camada que controla o acesso à memória (FTL), quando se utiliza o JFFS2, as chamadas são feitas diretamente para o MTD driver e, por fim, é feito o acesso à memória. Nesse cenário, é possível visualizar que os sistemas que não possuem FTL fazem acesso direto ao MTD 1.

Figura 2.11 | Sistema de memória flash para o sistema baseado em Linux

Fonte: adaptada de Deng (2017, p. 1).

Existe uma grande diferença entre sistemas embarcados baseados em placas prontas (*BeagleBoard* ou *Raspberry Pi*) e sistemas em que é necessário desenvolver o próprio hardware. Os custos e os conhecimentos necessários são diferentes, o custo de utilizar uma placa pronta pode ser maior, porém, o tempo de desenvolvimento é menor comparado ao sistema que deve montar o hardware. Com relação aos custos e à memória, como avaliar qual tipo de memória podemos utilizar em um sistema embarcado dado o processo de desenvolvimento do hardware?

O acesso a dispositivos MTD é feito por arquivos do tipo /dev/mtdX, onde o X representa qual dos sistemas de arquivo será acessado. Mesmo sendo um sistema de *raw flash*, ainda é necessário informar qual o sistema de arquivos que será utilizado. Os mtdX apresentam os dispositivos, em que cada partição representa uma nova entrada no /dev, mtdrX é o mesmo dispositivo, porém, em modo de apenas leitura, os mtdblockX são os dispositivos de bloco controlados pelo MTD, nftl**LN** representa os L dispositivos FTL com suas N partições.

O *Journalling Flash File System version* 2 (JFFS2) é um sistema de arquivos dedicado para memórias flash do tipo NOR e ainda pode ser utilizado para memórias do tipo not-and NAND (WOODHOUSE, 2011), sendo um sistema compactado e possuindo diversos elementos para evitar que as mesmas áreas da memória flash sejam utilizadas constantemente (as memórias flash possuem um limite de leitura e escritas) e ainda possui sistema de compactação (SALLY, 2010). Em sistemas GNU/Linux, podemos mapear um dispositivo com o JFFS2 com o comando mount -tjffs2 mtd0 /mnt informando qual é o sistema de arquivo (-tjffs2), o dispositivo (mtd0) e o ponto de montagem mnt (YAGHMOUR, 2013). O *Yet Another Flash File System* (YAFFS) é outro sistema de arquivos desenhado para sistema de memória flash NAND e sem compactação, sendo mais rápido e mais simples que o JFFS2 (SALLY, 2010). O YAFFS já foi utilizado pela NASA no projeto *Transiting Exoplanet Survey Satellite* (TESS) como parte de seu sistema embarcado.

Além do sistema de arquivos necessários para armazenamento, em alguns casos, a própria memória flash de armazenamento é utilizada como memória principal.

Familiarizar-se com os tipos de memória disponível e seus preços é de grande valia para um projeto de um sistema embarcado, podendo-se consultar os preços e os modelos disponíveis no link a seguir. Disponível em: <https://www.digikey.com/products/en/integrated-circuits-ics/memory/774>. Acesso em: 23 nov. 17.

Sem medo de errar

O projeto consiste na produção de um dispositivo que consiga medir o nível de água em um bueiro. Para isso, o sistema deve conter diversos sensores de nível para mensurar quão forte é a chuva. Nesse cenário, o problema consiste em escolher qual memória o sistema utilizará, se um dispositivo de armazenamento com a FTL ou sem essa camada.

Para apresentar a solução desse problema, deve-se montar uma tabela com as diferenças entre os dois tipos de memória:

	Com FTL	Sem FTL
Acesso para armazenamento	Feito de forma direta, apenas utilizando um sistema de arquivos clássico, como o ext3, FAT32 etc.	Necessário uma interface no sistema operacional e sistema de arquivos especializado.
Utilização pela aplicação	Pode fazer acesso direto pelo dispositivo mapeado pelo SO.	Deve fazer acesso pela interface do SO que faz o mapeamento.
Mapeamento físico	Feito pelo próprio chip.	Feito pelo SO.
Custo por bit	Maior.	Menor.
Tamanho físico	Maior.	Menor.
Sistema de arquivos	FAT32, ext3.	JFFS2, YAFFS.
Exemplo	Um sistema que utiliza o sistema de arquivos ext3 com FTL é Raspberry PI, com um micro SD *High Capacity* (MicroSDHC).	Nesse cenário, diversas memórias flash RAW em chips podem ser utilizadas. Para um cenário completo, pode-se usar um *target* com o FriendlyARM Mini2440, com 1 GB NAND Flash.

Avançando na prática

Medidor de qualidade elétrica residencial

Descrição da situação-problema

A cidade de São João está cansada de sofrer com problemas de eletricidade, o prefeito tem diversas reclamações dos moradores de sua cidade pela energia elétrica possuir variações na tensão, interrupções e ruídos. Diversos moradores já tiveram eletrodomésticos com avarias devido a esse problema elétrico. O prefeito entrou em contato com o seu primo, engenheiro eletricista, que indicou a compra de um equipamento que pudesse monitorar e guardar a data e a hora dos

eventos ocorridos, mas que esse equipamento pudesse ser instalado nas casas das pessoas.

O prefeito fez o pedido da construção desse equipamento para a equipe do seu primo, o produto deve monitorar e guardar qualquer variação na rede elétrica, todavia, deve ter baixo custo. Você faz parte dessa equipe de desenvolvimento e apresentará qual sistema de memória de armazenamento deverá ser utilizado, levando em conta:

- Baixo custo.
- Grande espaço de armazenamento.
- Resiliência para intempéries.

Resolução da situação-problema

A melhor opção para esse sistema seria uma memória do *raw flash* (sem a FTL). Com isso, é possível diminuir os custos e aumentar o tamanho de armazenamento e, ainda, como são elementos de armazenamento dedicados para sistemas embarcados, é possível encontrar elementos que possuem melhor proteção para as variações climáticas.

Faça valer a pena

1. Em sistemas embarcados, o espaço de armazenamento deve ser utilizado de forma coesa pela sua limitação física e lógica. Para isso, é necessário utilizar memórias que não sejam voláteis e possam ser aplicadas em ambientes ruidosos e com vibrações.

Qual das memórias listadas a seguir é mais indicada em ambientes com muita vibração?

a) Disco rígido de 15 RPM.

b) Memória flash do tipo NAND.

c) Memória RAM feita com capacitores.

d) Memória baseada em resistores.

e) Disco rígido do tipo SATA.

2. O projeto de um sistema embarcado tem vários pontos que devem ser levados em questão, por exemplo, a memória. Memórias para sistemas embarcados baseadas em *flash* podem ser *flash translation layer* (FTL) ou

serem do tipo *flash raw*. A escolha dos tipos de memória pode ser levada por diversas razões, tais como custo por bit e tamanho físico.

Sobre os tipos de memória *flash*, analise as afirmações:

I - A FTL agrega complexidade e criptografia.

II - Memórias do tipo *raw flash* devem ser utilizadas com capacitores.

III - Sem a FTL, é necessário utilizar um software junto ao sistema operacional.

IV - A FTL provê acesso aos dispositivos *flash raw*.

a) I e II estão corretas.

b) I, II e III estão corretas.

c) Apenas a I está correta.

d) II, III e IV estão corretas.

e) III e IV estão corretas.

3. Para diferentes dispositivos de armazenamento, é necessário um sistema de arquivos específico. Em sistema com *flash translation layer* (FTL), podemos utilizar sistemas de arquivos clássicos, como o ext3 e ext4. No entanto, para sistema sem a FTL, é necessário um sistema de arquivos relacionado ao tipo de circuito utilizado na construção da memória.

Dos sistemas de arquivos a seguir, marque quais são dedicados para sistema de *raw flash*:

a) JFFS2 e YAFFS.

b) ext4 e ReiserFS.

c) FAT32 e FAT16.

d) NTFS.

e) XFS.

Referências

BELL, C. **Windows 10 for the Internet of Things**. S.l.: Apress, 2016.

BORYCKI, D. **Programming for the internet of things**: Using Windows 10 IoT Core and Azure IoT Suite. S.l.: Microsoft Press, 2017.

CAPELLI, A. **Eletroeletrônica automotiva**: injeção eletrônica, arquitetura do motor e sistemas embarcados. São Paulo: Érica, 2010.

CHUNG, T. et al. A survey of flash translation layer. **Journal of Systems Architecture**, S.l., v. 1, n. 55, p. 332-343, fev. 2009.

DENG, S. Improving custom system using spansion linux flash file system. **International Conference on Consumer Electronics, Communications and Networks** (CECNet), S.l., p. 1853-1856, jan./nov. 2017.

GANSSLE, J. **The art of designing embedded systems**. Baltimore: Elsevier, 1999.

GANSSLE, J. **The art of designing embedded systems**. S.l.: Elsevier Science, 2008.

GANSSLE, J. G.; BARR, M. **Embedded systems dictionary**. Abingdon: Taylor & Francis, 2003.

MAUERER, W. **Professional Linux Kernel Architecture**. S.l.: John Wiley & Sons, 2010.

MITCHELL, M.; OLDHAM, J.; SAMUEL, A. **Advanced Linux Programming**. S.l.: New Riders Publishing, 2001.

NEGUS, C. **Linux Bible 2010 Edition**: Boot Up to Ubuntu, Fedora, KNOPPIX, Debian, openSUSE, and 13 Other Distributions. S.l.: Wiley, 2010.

NOERGAARD, T. **Embedded systems architecture**: a comprehensive guide for engineers and programmers. S.l.: Newnes, 2012.

OLIVEIRA, A. de; ANDRADE, F. de. **Sistemas embarcados**: hardware e firmware na prática. 2. ed. São Paulo: Érica, 2010.

PHUNG, S. **Professional microsoft windows embedded CE 6.0**. Indianapolis: Wiley, 2008.

SALLY, G. **Pro linux embedded systems**. S.l.: Apress, 2010.

TANENBAUM, A. S. **Sistemas operacionais modernos**. São Paulo: Pearson, 2008.

WIND RIVER. **Wind river real-time core for linux**. 2009. Disponível em: <https://www.windriver.com/products/product-overviews/PO_RTCore_for_LX_May2009.pdf>. Acesso em: 21 nov. 2017.

WOODHOUSE, D. **JFFS**: the journalling flash file system. Ottawa Linux Symposium. S.l., nov. 2011.

YAGHMOUR, K. **Building embedded linux systems.** S.l.: O'Reilly Media, Inc., 2003.

______. **Embedded android**: porting, extending, and customizing. [s.l.]: O'Reilly Media, Inc., 2013.

ZHANGJIN, W.; ZIQIANG, C. **Instant optimizing embedded systems using BusyBox**. S.l.: ProQuest, 2013.

Configurações relacionadas aos sistemas embarcados

Convite ao estudo

Prezados alunos, nesta unidade serão apresentados os conceitos práticos para o desenvolvimento de um sistema embarcado. Neste momento do estudo nos aprofundaremos no processo de desenvolvimento de um sistema embarcado, dessa forma, os processos serão mais intricados.

Nesta unidade será possível resolver problemas mais complexos e precisos de um sistema embarcado. Problemas como sistemas de monitoramento de ambientes críticos, cenários em ambientes para *Internet of Things* (IoT) e para sistemas de controle de acesso. Todavia, nessa etapa resolveremos problemas mais específicos, por exemplo, definir como o sistema fará sua inicialização ou a comunicação via rede.

Para que seja possível o desenvolvimento de um sistema embarcado, passaremos pelas seguintes etapas (ZHANGJIN, 2013):

1. Criação do root *filesystem* – sistema de arquivos no qual os dados do sistema operacional (SO) são armazenados.
2. Configuração e compilação das ferramentas do sistema operacional – criaremos todos os comandos e funções para utilizar o hardware.
3. Configuração e compilação do *kernel* – o *kernel* é o software que conversa diretamente com o hardware.
4. Formação do sistema de arquivo – definir como será arquitetada a memória de armazenamento do sistema embarcado.

5. Instalação do bootloader – software que faz a inicialização e configuração inicial do sistema de arquivos e inicializa o kernel.
6. Instalação da aplicação foco do sistema embarcado – inserir no *firmware* do sistema embarcado quais serão as funções que darão a utilização do sistema embarcado.
7. Envio do *firmware* para o sistema target - o *firmware* é o software enviado para a placa que executará o sistema embarcado (target).

Neste momento, aplicaremos diversos conceitos que já foram estudados e será necessário entrar no mundo dos sistemas GNU/Linux, que em um primeiro momento podem parecer um pouco diferente do Microsoft Windows, que normalmente é mais utilizado. Todavia, com um pouco de persistência e análise crítica será possível utilizar essas ferramentas sem problemas.

As vantagens de um sistema GNU/Linux em relação ao desenvolvimento, acesso a ferramentas e dinamismo na utilização serão visíveis rapidamente.

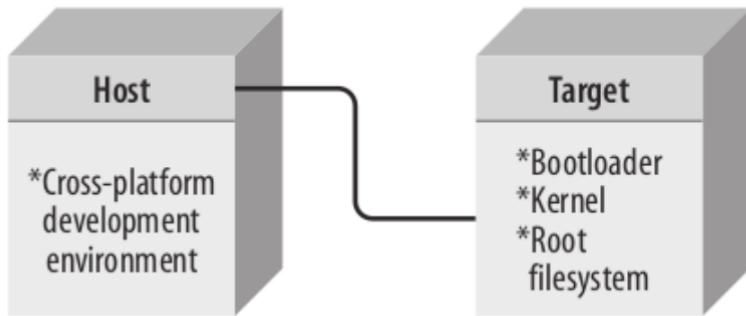
Seção 3.1

Configuração de sistemas de arquivos embarcados

Diálogo aberto

Uma equipe de quatro pessoas formadas em Engenharia Química criaram, com o uso de técnicas de dinâmica molecular (técnica que utiliza o computador para simular átomos e moléculas), um composto químico capaz de transformar plástico em dióxido de carbono (CO_2) . Agora, a segunda etapa do projeto consiste em sintetizar esse composto e fazer testes em laboratório. Todavia, o líder do projeto teme que nos testes de laboratório a concentração de CO_2 fique muito alta ou ainda seja liberado algum outro tipo de gás, em ambas as situações a equipe de testes pode correr riscos.

Para resolver esse problema, você foi chamado para elaborar um guia com os procedimentos necessários para criar um sistema embarcado das etapas de criação do sistema de arquivos e configuração das ferramentas do sistema operacional. Com esse guia a equipe de testes de laboratório pode iniciar o projeto do sistema embarcado. Esse sistema fará a leitura dos gases do ar e se algum limite for ultrapassado um aviso sonoro deve ser emitido. Os procedimentos não devem apresentar todos os detalhes dos comandos, e sim quais são os passos e quais comandos do GNU/Linux devem ser utilizados.

Não pode faltar

O desenvolvimento de um sistema embarcado é um processo que envolve diversas ferramentas, a utilização de um sistema como base para a geração do código e o envio desse código para a plataforma-alvo (OLIVEIRA, 2010). Conforme a Figura 3.1, o host é o computador em que é feito o desenvolvimento do sistema embarcado e deve conter as ferramentas de desenvolvimento; o *target* é o equipamento que receberá o *firmware* (software incorporado ao hardware) (GANSSLE, 2001). O *target* tem 3 elementos básicos para que o sistema possa inicializar, são eles: *bootloader*, *kernel* e *root filesystem*,

O *bootloader* é responsável por iniciar o carregamento do sistema, o *kernel*, que foi carregado pelo *bootloader*, provê acesso ao hardware e o root *filesystem* é a estrutura de armazenamento em que os dados, bibliotecas e aplicações são guardados (YAGHMOUR, 2013).

Segundo Zhangjin (2013), o desenvolvimento de um sistema embarcado pode se dividir nas seguintes etapas:

1. Criação do root *filesystem*.
2. Configuração e compilação das ferramentas do sistema operacional.
3. Configuração e compilação do *kernel*.
4. Formação do sistema de arquivo.
5. Instalação do *bootloader*.
6. Instalação da aplicação foco do sistema embarcado.
7. Envio do *firmware* para o sistema *target*.

Figura 3.1 | Forma de desenvolvimento de um sistema embarcado com sistema operacional

Fonte: Yaghmour (2003, p. 40).

Vale lembrar que o SO, em uma visão geral, é composto por dois grandes elementos: o *kernel*, que faz a interface entre o hardware e as outras camadas do SO, e a outra grande camada são as ferramentas e aplicações que usam o *kernel* (TANENBAUM, 2008). Para aplicações embarcadas, a configuração de um SO não é diferente, todavia questões de espaço são extremante relevantes, para isso, pode-se utilizar o BusyBox.

Segundo Zhangjin (2013), o BusyBox foi criado para ser uma pequena distribuição utilizada na instalação do Debian (distribuição de GNU/Linux), devendo ser pequeno o suficiente para ser instalado em um *floppy disk*. Porém, essa característica de ser compacto acabou levando a sua utilização em sistemas embarcados. O BusyBox consiste em um kit de desenvolvimento que provê diversas ferramentas do GNU/Linux. Para que seja possível utilizar esse kit, o primeiro passo consiste em configurar o root *filesystem*.

O *root filesystem* consiste na estrutura de diretórios e na definição do sistema de arquivos (ext3, FAT e outros) que serão utilizados. Porém, para que seja possível fazer essa configuração, é necessário configurar o host com as ferramentas de desenvolvimento. Com isso, tomamos como base um sistema GNU/Linux Ubuntu, que terá o papel de *host*; para outras distribuições os comandos poderão ser diferentes, mas o processo é o mesmo. Utilizaremos o *apt-get* que é uma ferramenta para instalação de pacotes no Ubuntu, o comando abaixo faz a instalação de todas as ferramentas necessárias:

- sudo apt-get install gcc g++ make build-essential gcc-arm-linux-gnueabi e2fslibs ia32-libs ia32-libs-multiarch.

Esses pacotes são responsáveis por fornecer as ferramentas descritas no Quadro 3.1.

Quadro 3.1 | Função de cada pacote

Pacote	Função
gcc	Fornece o compilador para a linguagem C do *GNU Compiler Collection* (GCC).
g++	Fornece o compilador para a linguagem C++ do *GNU Compiler Collection* (GCC).
make	Utilitário para compilação dos softwares.
build-essential	Pacote que instala gcc, g++ e bibliotecas de sistema.
gcc-arm-linux-gnueabi	Compilador em c para arquitetura Advanced RISC Machine (ARM); esta é a arquitetura de diversos sistemas embarcados.
ia32-libs	Bibliotecas para sistema 32 bits x 86.
ia32-libs-multiarch	Bibliotecas para sistema 32 bits x 86.
e2fslibs	Ferramentas para o sistema de arquivos.

Fonte: elaborado pelo autor.

Com as ferramentas de desenvolvimento instaladas é possível iniciar o processo de criação do sistema de arquivos do sistema embarcado. Como primeiro passo e para manter a organização no host é necessário criar a seguinte estrutura de diretórios:

- *nomeDoSistemaEmbarcado*
  - *src.*
  - *Build.*
  - *Target.*

No diretório *src* ficarão todos os códigos-fonte dos elementos de software que serão utilizados no projeto. Na pasta *build* ficarão todos os arquivos derivados do processo de compilação, e na pasta *target* toda a estrutura que será enviada para o sistema *target*.

Dentro do diretório *target* é necessário criar com o comando *mkdir* todos os diretórios necessários que serão utilizados no sistema embarcado, tais como (YAGHMOUR, 2003):

- dev
  - Diretório que contém as referências (arquivos) que representam os dispositivos.
- dev/pts
  - Contém os arquivos que representam os terminais.
- etc
  - Guarda os arquivos de configuração do sistema.
- etc/init.d
  - Possui os arquivos de configuração de inicialização dos sistemas.
- lib
  - Bibliotecas principais para o projeto embarcado.
- mnt
  - Diretório no qual ficam mapeados dispositivos de armazenamento.
- opt
  - Local de instalação de softwares do sistema.
- proc
  - Diretório onde o *kernel* mapeia informações.
- root

- sys
  - Diretório onde o *kernel* mapeia informações.
- tmp
  - Arquivos temporários.
- var
  - Armazenamento de *logs* e cache.
- var/log
  - Armazenamento de *logs* do sistema.

A Figura 3.2 apresenta como é realizado o processo de criação dos diretórios do sistema de arquivos utilizando o terminal do Debian.

Figura 3.2 | Criação de diretórios do sistema de arquivos

```
>  mkdir -p dev dev/pts etc etc/init.d lib mnt opt proc root sys tmp var var/log
> ls
dev  etc  lib  mnt  opt  proc  root  sys  tmp  var
> 
```

Fonte: elaborada pelo autor.

Um outro elemento necessário é a criação de alguns dispositivos básicos do sistema *target*. O comando *mknod* cria dispositivos para o sistema *target*, com o comando (NEGUS, 2010):

- mknod dev/console c 5 1: a primeira opção dev/console informa o nome do dispositivo; a opção c informa para criar um sistema de caracteres; o 5 e 1 são números para identificação do dispositivo.

Dentro de cada diretório serão inseridos diversos elementos de configuração para possibilitar a utilização do sistema. Todavia, os detalhes desses arquivos serão apresentados nas próximas etapas, juntamente com as ferramentas e o *kernel*. Quando todas essas etapas forem realizadas, pode-se criar o arquivo com o *firmware*, para isso, deve-se (YAGHMOUR, 2003:

1. Criar um arquivo especial que servirá como sistema de armazenamento.
2. Formatar esse arquivo com o sistema de arquivos desejado.
3. Mapear esse arquivo para seja possível copiar os arquivos que estão no diretório *target*.

4. Copiar os arquivos do diretório *target* para o diretório mapeado.

Para esse conjunto de tarefas, com base em um sistema Debian, é possível utilizar os comandos (NEGUS, 2010):

- dd: utilitário para cópia e conversão de arquivos. A cópia, nesta etapa, acontece do /dev/zero (dispositivos que fornece 0) para o arquivo fs.bin, de tamanho *count* multiplicado pelo block size (bs). A Figura 3.3 apresenta o resultado da criação dessa imagem inicial.
    - dd if=/dev/zero of=fs.bin count=10000 bs=1k.

Figura 3.3 | Criação da imagem inicial

```
> dd if=/dev/zero of=fs.bin count=10000 bs=1k
10000+0 records in
10000+0 records out
10240000 bytes (10 MB, 9.8 MiB) copied, 0.0167916 s, 610 MB/s
> █
```

Fonte: elaborada pelo autor.

- mke2fs: comando que cria o sistema de arquivo do tipo *extended filesystem*. A opção -F força a criação do sistema de arquivos, e a opção -m0 tem a função de evitar a reserva de blocos/segmentos do sistema de arquivos para o usuário administrador do sistema. A Figura 3.4 apresenta o resultado da criação do sistema de arquivos.
    - mke2fs -F -m0 fs.bin

Figura 3.4 | Criação do sistema de arquivos da imagem

```
>  mke2fs -F -m0 fs.bin
mke2fs 1.43.4 (31-Jan-2017)
Discarding device blocks: done
Creating filesystem with 10000 1k blocks and 2512 inodes
Filesystem UUID: 43e4f48a-d8ac-4c12-a467-729a3d77fe95
Superblock backups stored on blocks:
        8193

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

> █
```

Fonte: elaborada pelo autor.

- mount: ferramenta que mapeia o sistema de armazenamento em diretórios. A opção -t força o mapeamento utilizando o sistema de arquivos ext2 na opção /mnt, e a opção -o informa que o arquivo é utilizado como dispositivo de armazenamento.
  - mount -t ext2 fs.bin /mnt -o loop
- cp: copia arquivos. A opção v mostra quais arquivos foram copiados, a opção r faz a cópia de todos os subdiretórios e f copia sem considerar avisos.
  - cp -vfr target /mn
- umount: remove o mapeamento feito pelo mount.
  - O umount remove o mapeamento do /mnt e força a escrita de qualquer *buffer* temporário.

Para a criação do sistema de arquivo raiz ou *root file system* é necessário criar um arquivo que servirá como dispositivo de armazenamento em um processo semelhante àquele em que utiliza um sistema de virtualização como o *VirtualBox*. Depois disso, é necessário criar o sistema de arquivos de forma parecida a de um pen *drive* ou cartão SD. Por fim, enviar os arquivos para esse dispositivo.

Com a construção do sistema de arquivos é possível iniciar o processo de configuração das ferramentas de sistema operacional, para isso, é necessário buscar o código fonte do BusyBox. É possível fazer o download do *wget* em: <https://busybox.net/downloads/busybox-1.27.2.tar.bz2>. Acesso em: 6 mar. 2018.

Com o *wget* no terminal é possível recuperar o código-fonte do código. Na Figura 3.5 temos um exemplo de um terminal do Ubuntu, criando uma pasta com o comando *mkdir* e fazendo o download do código-fonte do BusyBox e sua descompactação com o comando *tar* e as opções *xf*.

Figura 3.5 | Download do código-fonte e descompactação do BusyBox

```
File  Edit  View  Search  Terminal  Help
> mkdir busybox
> cd busybox
> wget https://busybox.net/downloads/busybox-1.27.2.tar.bz2
--2017-11-19 12:38:04--  https://busybox.net/downloads/busybox-1.27.2.tar.bz2
Resolving busybox.net (busybox.net)... 140.211.167.122
Connecting to busybox.net (busybox.net)|140.211.167.122|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2216527 (2.1M) [application/x-bzip2]
Saving to: ‘busybox-1.27.2.tar.bz2’

busybox-1.27.2.tar.b 100%[=====================>]   2.11M   295KB/s    in 7.7s

2017-11-19 12:38:12 (281 KB/s) - ‘busybox-1.27.2.tar.bz2’ saved [2216527/2216527]

> tar -xf busybox-1.27.2.tar.bz2
> 
```

Fonte: elaborada pelo autor.

Exemplificando

Um outro exemplo de sistema operacional embarcado é o *Wind River* ou também conhecido como *Yocto Project* (Disponível em: <https://www.yoctoproject.org/>. Acesso em: 6 mar. 2018). Com ele é possível criar um sistema baseado em modelos e, com isso, criar uma distribuição de Linux embarcado dedicado para o hardware alvo.

A arquitetura do *target* pode não ser a mesma do *host*, por essa razão instalou-se compiladores para outras arquiteturas (ARM). Após executar a descompactação é necessário utilizar o comando *make* (ferramenta para auxiliar no processo de compilação), com as opções ARCH para indicar outra arquitetura e a opção CROSS_COMPILE para indicar qual compilador utilizar. A opção final do comando *make* indica o que será compilado (ZHANGJIN, 2013):

- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-defconfig.
- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-menuconfig.

A opção *defconfig* executa uma configuração inicial padrão para a compilação do BusyBox, a Figura 3.6 apresenta o resultado dessa configuração básica.

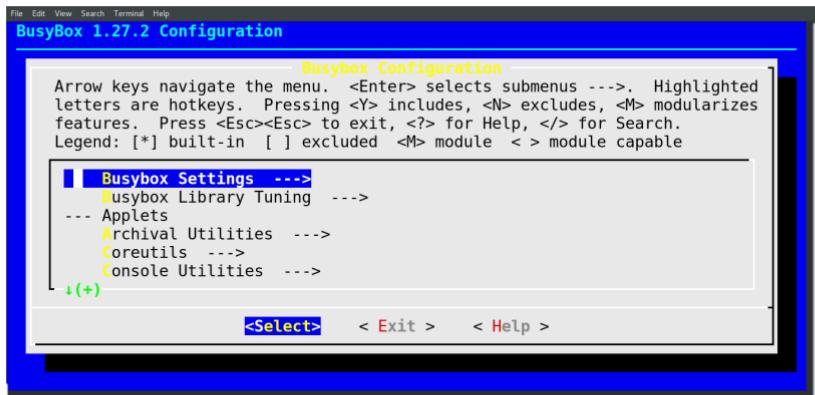
Figura 3.6 | Resultado da configuração padrão para o BusyBox

```
File  Edit  View  Search  Terminal  Help
*
klogd (KLOGD) [Y/n/?] (NEW) y
  *
  * klogd should not be used together with syslog to kernel printk buffer
  *
  Use the klogctl() interface (FEATURE_KLOGD_KLOGCTL) [Y/n/?] (NEW) y
logger (LOGGER) [Y/n/?] (NEW) y
logread (LOGREAD) [Y/n/?] (NEW) y
  Double buffering (FEATURE_LOGREAD_REDUCED_LOCKING) [Y/n/?] (NEW) y
syslogd (SYSLOGD) [Y/n/?] (NEW) y
  Rotate message files (FEATURE_ROTATE_LOGFILE) [Y/n/?] (NEW) y
  Remote Log support (FEATURE_REMOTE_LOG) [Y/n/?] (NEW) y
  Support -D (drop dups) option (FEATURE_SYSLOGD_DUP) [Y/n/?] (NEW) y
  Support syslog.conf (FEATURE_SYSLOGD_CFG) [Y/n/?] (NEW) y
  Read buffer size in bytes (FEATURE_SYSLOGD_READ_BUFFER_SIZE) [256] (NEW) 256
  Circular Buffer support (FEATURE_IPC_SYSLOG) [Y/n/?] (NEW) y
    Circular buffer size in Kbytes (minimum 4KB) (FEATURE_IPC_SYSLOG_BUFFER_SIZE) [1
6] (NEW) 16
  Linux kernel printk buffer support (FEATURE_KMSG_SYSLOG) [Y/n/?] (NEW) y
>
```

Fonte: elaborada pelo autor.

Reflita

Um sistema baseado em BusyBox pode ser modelado para ter os recursos necessários dado o hardware que será utilizado no sistema embarcado. Para fazer que seja utilizado o menor espaço possível é necessário compilar o BusyBox e o *kernel* do Linux com o mínimo de recursos. Porém, será que o tempo para remover elementos do SO é maior do que o tempo para projetar um hardware com mais memória?

Se no final do comando *make* for utilizada a opção *menuconfig* será apresentada uma interface que apresenta diversas configurações, sendo que cada opção selecionada insere novos recursos que podem ser utilizados no BusyBox (ZHANGJIN, 2013). A Figura 3.7 apresenta uma parte dessa interface. Essa etapa representa a configuração dos comandos, *drivers* e módulos que estão disponíveis para uso no sistema operacional embarcado. Quanto mais recursos forem adicionados, mais espaço será utilizado. Nesta etapa deve-se levar em conta o hardware que será utilizado e quais recursos serão necessários para que a aplicação a ser embarcada possa funcionar.

O Quadro 3.2 a seguir apresenta algumas opções que são úteis.

Quadro 3.2 | Descrição das opções para configuração do *kernel*

Funcionalidade	*Caminho para configuração pelo menuconfig*
Sistemas de arquivos	*Linux System Utilities →* *FileSystem/Volume identification →* ** jfs filesystem* ** xfs filesystem* ** Ext filesystem* ** fat filesystem*
Comandos básicos	*Coreutils →* ** dd* ** df* ** ls*
Comandos auxiliares para console	*Console Utilities →* ** setconsole* ** showkeys* ** setfont*
Editores	*Editors →* ** awk* ** vi* ** diff*

Fonte: elaborado pelo autor.

Figura 3.7 | Interface de configuração do BusyBox para a seleção de funcionalidades

```
File  Edit  View  Search  Terminal  Help
BusyBox 1.27.2 Configuration
                    Busybox Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted
  letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

        Busybox Settings  --->
        Busybox Library Tuning  --->
    --- Applets
        Archival Utilities  --->
        Coreutils  --->
        Console Utilities  --->
    ↓(+)

            <Select>    < Exit >    < Help >
```

Fonte: elaborada pelo autor.

Em configurações padrões do BusyBox obtemos um conjunto de comandos. Os mais utilizados são:

- adduser – adiciona usuários.
- awk – formata a saída de texto.
- bzip2 – compactador de arquivos.
- cat – lista arquivos.
- chmod – muda permissão de diretórios e arquivos.
- chown – muda proprietário de diretórios e arquivos.
- deluser – remove usuários.
- free – lista utilização da memória de trabalho do sistema.
- fsck – formata dispositivos de armazenamento.
- grep - filtra textos por padrões.
- ifconfig – configura interface de redes.
- mkdir – cria diretórios.
- passwd – altera senhas.

Em Zhangjin (2013) existe uma lista com todos os comandos disponíveis no BusyBox. Com o BusyBox pronto, o próximo passo consiste na configuração do kernel e dos *bootloaders*, assim o *firmware* (composto pelo *kernel*, *bootloader* e BusyBox) será enviado para o hardware *target*. Esse envio pode ser feito por uma comunicação via USB, serial ou gravando-a em uma memória SD.

Pesquise mais

Alguns elementos para novos conhecimentos de sistema embarcado podem ser encontrados em:

- <https://www.yoctoproject.org>. Acesso em: 4 dez. 2017.
- <https://www.youtube.com/watch?v=Sk9TatW9ino>. Acesso em: 4 dez. 2017.
- <https://busybox.net/about.html>. Acesso em: 4 dez. 2017.

Para sistemas baseados em Windows utiliza-se o Windows 10 IoT. Alguns links com informações:

- <https://www.youtube.com/watch?v=Ie3ICecqG3g>. Acesso em: 4 dez. 2017.

- <https://developer.microsoft.com/pt-br/windows/iot>. Acesso em: 4 dez. 2017.
- BELL, C. **Windows 10 for the Internet of Things.** [S.l.]: Apress, 2016.
- <https://msdn.microsoft.com/pt-br/magazine/mt808503.aspx>. Acesso em: 4 dez. 2017.

Sem medo de errar

Neste cenário é necessário elaborar um guia de como criar um sistema embarcado que fará a leitura dos gases do ar e que precisará, se algum limite for ultrapassado, emitir um aviso sonoro. Esse guia deve apresentar os passos principais para o desenvolvimento do sistema e os comandos utilizados.

- Criação do root filesystem:
  - *mkdir*: criação de diretórios.
  - *mknode*: criação de dispositivos.
- Configuração e compilação das ferramentas do sistema operacional:
  - *wget*: download dos arquivos com as ferramentas, tal como o BusyBox.
  - *tar*: descompactação dos arquivos.
  - *mkdir*: criação de diretórios.
  - *make*: compilação das ferramentas.
- Formação do sistema de arquivo:
  - *dd*: copia dados de dispositivos e arquivos.
  - *mke2fs*: cria um sistema de arquivos do tipo ext2.
  - *mount*: mapeia um dispositivo de armazenamento em um diretório.
  - *cp*: copia arquivos.
  - *umount*: remove mapeamento de um diretório vindo de um dispositivo.

Avançando na prática

Alimentador de gatos

Descrição da situação-problema

A empresa petFeed está desenvolvendo um alimentador para gatos, que consiste em um contêiner para armazenar o alimento, uma portinhola que faz o alimento cair na vasilha do gato e um sistema de alto-falantes e microfone para que os donos possam gravar mensagens para seus felinos. Toda vez que o alimento é liberado uma mensagem é tocada no alto-falante. Esse sistema é controlado por um sistema embarcado baseado em BusyBox, porém os consumidores estão reclamando que não conseguem gravar mensagens suficientes no alimentador. Seu papel é indicar para a petFeed como liberar memória do BusyBox para que seja possível gravar mais mensagens.

Resolução da situação-problema

No caso apresentado é preciso buscar quais elementos não são necessários no *firmware* de produção do produto. Arquivos de *logs* excessivos, ferramentas que não são utilizadas ou serviços que não são aplicáveis no *firmware* de produção podem ser removido sem causar problemas. Dessa forma, para resolver essa situação é necessário recompilar todo o sistema embarcado, porém, ao refazer a compilação, deve-se desmarcar as opções que não são utilizadas. Para refazer a compilação do BusyBox:

- make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- menuconfig.

Ao entrar no menu podem ser removidas algumas opções, como as descritas no Quadro 3.3.

Quadro 3.3 | Descrição dos elementos que podem ser alterados no BusyBox

Funcionalidade	Caminho para configuração pelo *menuconfig*
Sistemas de arquivos	*Linux System Utilities* → *FileSystem/Volume identification* → Remover todas os sistemas de arquivos que não serão utilizados.
Comandos auxiliares para console	*Console Utilities* → Remover todos os comandos que não serão utilizados.
Editores	*Editors* → Para a produção, raramente se utilizam editores, assim, todos podem ser removidos.

Fonte: elaborado pelo autor.

Faça valer a pena

1. O BusyBox é constituído por um conjunto de ferramentas feito pela GNU para funcionar com o *kernel* do tipo UNIX. Ao unir um *kernel* com o Linux um sistema operacional para um sistema embarcado é criado. De diversas características de um sistema operacional para um sistema embarcado, duas podem ser evidenciadas: tamanho reduzido e capacidade de receber novos softwares.

Além das ferramentas fornecidas pelo BusyBox e *kernel*, quais componentes são necessários para tornar o projeto embarcado funcional?

a) Sistema de arquivo raiz e *bootloader*.
b) Sistema de arquivo.
c) Ferramentas de depuração e atuadores.
d) Atuadores e módulos.
e) Drivers.

2. O sistema de arquivos raiz tem papel importante no sistema embarcado, com ele é possível armazenar todos os arquivos, as aplicações e o *kernel*. A criação de um sistema de arquivo raiz baseado em GNU/Linux deve ter no mínimo 4 passos que envolvem criação e cópias de arquivos.

Para a criação de um sistema de arquivo do tipo ext2, deve-se utilizar o comando:

a) mkdir.
b) mknod.
c) mke2fs.
d) df.
e) ls.

3. Para o desenvolvimento de um sistema embarcado são necessárias diversas etapas:

1. Criação do *root filesystem*.
2. Configuração e compilação das ferramentas do sistema operacional.
3. Configuração e compilação do *kernel*.
4. Formação do sistema de arquivo.
5. Instalação do *bootloader*.
6. Instalação da aplicação foco do sistema embarcado.
7. Envio do *firmware* para o sistema *target*.

Das sete etapas, em quais delas é possível reduzir a imagem que será enviada para o sistema *target* utilizando as opções e removendo os itens que não serão utilizados no produto final?

a) Etapas 1 e 7.
b) Etapas 1, 2, 3 e 4.
c) Etapas 6 e 7.
d) Etapas 4, 5 e 6.
e) Etapas 1, 2, 3, 4, 5, 6 e 7.

Seção 3.2

Carregador de inicialização de sistemas embarcados

Diálogo aberto

Na etapa de estudos anterior, abordamos a criação de um ambiente de desenvolvimento para sistemas embarcados, bem como a criação dos sistemas de arquivos e estrutura de diretórios. Nesta etapa, o foco será os *bootloaders* (parte do *firmware* que cuida da inicialização do sistema embarcado) e as configurações pertinentes para cenários diferentes de aplicação.

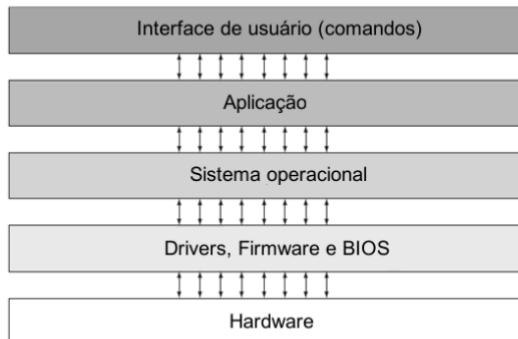
Os projetos da empresa KtechPi estão cada vez mais elaborados, hoje o líder da sua equipe apresentou as novas fronteiras que a empresa quer chegar. O novo projeto da empresa consiste em uma rede elaborada de sensores espalhados na cidade para maximizar a qualidade de vida das pessoas. Sensores de temperatura, umidade, velocidade do vento, radiação solar, qualidade do ar, intensidade luminosa, etc., que trarão informações em tempo real para que as pessoas possam ser avisadas de eventos extremos e planejar melhor seu dia a dia.

Dentro desse cenário, a nova demanda consiste em um sistema embarcado para monitoramento da qualidade do ar de diversos pontos da cidade, todavia, o sensor que será utilizado para fazer essa medição ainda é experimental e pode requerer atualizações constantes do software. Essas mudanças podem ser semanais, o que leva a necessidade de atualizar o *firmware* do sistema embarcado de forma constante. O sistema embarcado já será instalado em sua localização final, equipado com rede de fibra óptica.

Você foi escalado para projetar um sistema embarcado com sistema operacional para que seja feito esse monitoramento da qualidade do ar e garantir que seja possível a atualização do *firmware* de forma correta, sem correr riscos. Para isso, prepare uma apresentação de 5 minutos demostrando:

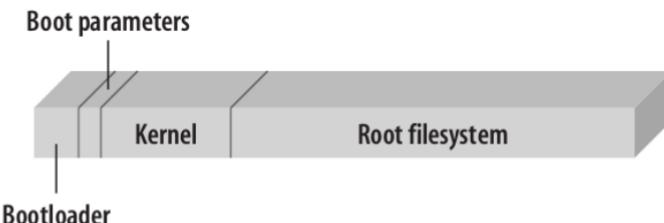
1. Os conceitos do processo de inicialização de um sistema embarcado.
2. O tipo de arquitetura utilizada.
3. Qual *bootloader* utilizar.

Não pode faltar

Um conjunto de circuitos eletrônicos como resistores e transistores necessitam de corrente e tensão elétrica para seu funcionamento, assim, cada um dos componentes pode permitir, bloquear ou modular a energia (OLIVEIRA; ANDRADE, 2010). Quando o foco é o computador, ao receber a energia elétrica, diversos componentes eletrônicos são inicializados e um circuito integrado (CI) em específico também começa suas operações (STALLINGS, 2009). Essa parte inicial é chamada de *basic input/output system* (BIOS), sendo que este circuito integrado tem o papel de inicializar e testar o hardware do computador (TANENBAUM, 2013). Além desse papel importante é necessário que o BIOS indique qual dispositivo tem instalado os arquivos referentes ao sistema operacional. Na Figura 3.8 temos uma visão esquemática de um computador no qual o hardware possui interface com os *firmwares* e o BIOS, dialogando com o sistema operacional que, por sua vez, faz a interface com a aplicação e desta com o usuário.

A tradução direta de *boot* é bota ou botina, o nome *boot* vem de *As aventuras do Barão de Munchausen*, de Rudolf Erich Raspe. As histórias contam que o Barão era capaz de evitar poças de água se erguendo ao puxar os cadarços da bota, levantando-se pela própria força. Faça uma pesquisa verificando a origem dessa história.

Hoje, algumas traduções já indicam *boot* como inicialização.

Figura 3.8 | Visão esquemática de um computador

Fonte: adaptado de Berger (2005, p. 4).

O BIOS em computadores de uso geral (computadores pessoais, servidores, notebooks e outros) possui diversos elementos de configuração de hardware. Essas configurações são essenciais para que o computador possa ser utilizado, todavia, em sistemas embarcados, temos uma parte mais simples que se chama *boot ROM*.

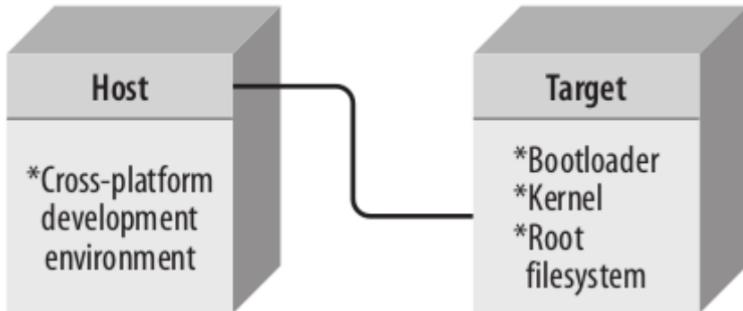
Segundo Ganssle (2008), a *boot ROM* tem papel essencial em um sistema embarcado, pois é comparável ao BIOS de computadores gerais. A boot ROM é responsável:

1. Pela inicialização de sinais de *clocks* (sinais para sincronismo de operações).
2. Pela detecção da mídia onde está armazenado o *boot loader* (parte do sistema operacional dedicada à inicialização do kernel).
3. Pela inicialização de memórias do tipo RAW para que seja possível acesso ao boot loader.
4. Pela configuração básica de sistema de arquivos para inicialização do sistema operacional.
5. Por utilizar drivers para acesso de dispositivos de armazenamento, tais como sistema de memória USB.

A *boot ROM* está presente nos sistemas embarcados como ponte de acesso ao sistema operacional. Essa ponte tem como papel principal indicar e proporcionar acesso para que o *bootloader* possa ser carregado para, então, utilizar os parâmetros de boot para iniciar o *kernel* e, por fim, o *root filesystem*. A Figura 3.9 apresenta como é arquitetada a memória de um sistema embarcado baseado em memória de estado sólido. Em primeiro lugar, o *bootloader* é armazenado, depois são os parâmetros de boot e, por fim, o *kernel* e o *root filesystem*.

Figura 3.9 | Memória não volátil de um sistema embarcado

Fonte: Yaghmour (2003, p. 49).

A *boot ROM* é um *firmware* que pertence à placa de desenvolvimento ou ao hardware que foi construído no projeto do sistema embarcado. Esta pequena porção de código tem o papel de fornecer acesso básico ao hardware.

O *bootloader* é um software que está na memória de armazenamento do sistema embarcado. Ele é direcionado pela *boot* ROM para ser executado e faz o processo de carregamento do *kernel*, que, por sua vez, faz o do sistema de arquivos.

Segundo Pavlov e Belevsky (2008) as funções de um *bootloader* são:

- Inicializar hardware.
- Inicializar a plataforma para que seja possível o carregamento da imagem do sistema operacional.
- Carregar para a memória de trabalho o sistema operacional.
- Inicializar o sistema operacional.

Segundo Yaghmour (2003), existem três cenários diferentes de aplicação em que as configurações do *bootloader* podem diferir. O cenário mais utilizado é o da Figura 3.10, no qual todas as ferramentas são utilizadas no computador, enquanto no *target* temos todo o sistema operacional embarcado (*bootloader*, *kernel* e *root filesystem*) gravado na memória do sistema de armazenamento do sistema embarcado. A *boot ROM* do hardware indica em qual dispositivo está armazenado o *bootloader* e, por sua vez, o *bootloader* faz a inicialização do sistema embarcado.

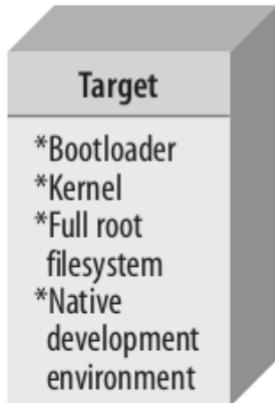



Figura 3.10 | Arquitetura de um sistema embarcado baseado em *host* e *target* local

Fonte: Yaghmour (2003 p. 39).

Outra arquitetura é apresentada na Figura 3.11, na qual todos os componentes de um sistema operacional embarcado (*bootloader*, *kernel*, *root filesystem* e kit de desenvolvimento) estão instalados diretamente no *target*.

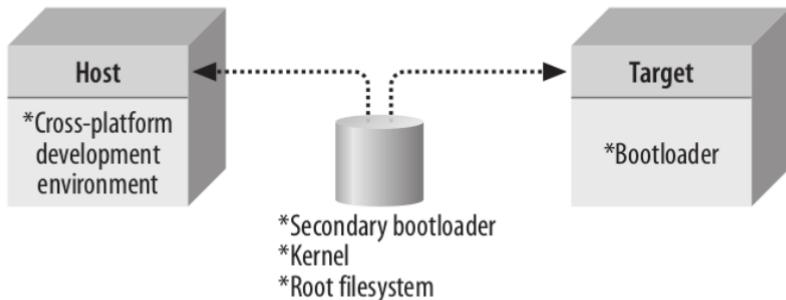



Figura 3.11 | Arquitetura de um sistema embarcado baseado apenas no *target*

Fonte: Yaghmour (2003, p. 41).

Por fim, a última configuração possível de um sistema embarcado é retratada na Figura 3.12, em que o *host* contém as ferramentas de desenvolvimento, e o *target* possui o *bootloader*. Quando o *target* é iniciado, busca um dispositivo de armazenamento remoto com um segundo *bootlader* que indicará qual *kernel* e qual sistema de arquivos carregar na memória do sistema embarcado.

Figura 3.12 | Arquitetura de um sistema embarcado baseado em host e *target* em que o armazenamento do *kernel* e root file system é feito de maneira remota

Fonte: Yaghmour (2003, p. 40).

Exemplificando

Um sistema baseado em *Raspberry Pi* utiliza uma estratégia de inicialização como na Figura 3.11, em que apenas host é necessário para a programação do software embarcado, com ele é possível instalar um sistema GNU/Linux completo com acesso à internet. Dessa forma, é possível fazer o desenvolvimento das aplicações, testes e validações sem a necessidade de um outro hardware com as ferramentas de desenvolvimento. Em um sistema baseado em um hardware menos amigável (por exemplo, o FriendlyARM Mini2440, em que é preciso instalar diversas ferramentas, drivers e programa para acesso) é necessário instalar um *bootloader* específico, tal como o U-Boot.

Os tipos de *bootloader* estão relacionados à arquitetura do *target* e suas necessidades. Em um ambiente baseado em X86, normalmente é utilizado o *GRand Unified Bootloader* (GRUB), pois ele pode ser utilizado nos três ambientes de um sistema embarcado (*target* e host local, host e *target* remoto e apenas *target*) (SALLY, 2010). Segundo Sally (2010), o GRUB possui funcionalidades para fazer o boot local ou pela rede por meio de protocolos: *Trivial File Transfer Protocol* (TFTP), *Bootstrap Protocol* (BOOTP) ou *Dynamic Host Configuration Protocol* (DHCP).

Porém, por questões de arquitetura e ferramentas de desenvolvimento, segundo Yaghmour (2003), o "Das U-Boot" (de origem alemã, que significa "o submarino") ou U-Boot é o *bootloader* mais utilizado quando se trata de arquiteturas do tipo Advanced RISC Machine (ARM), pois ARM é a arquitetura mais utilizada em sistema embarcados. Para utilizar o U-Boot é necessário buscar seu código-fonte, com o comando git é possível acessar a última versão.

A Figura 3.13 apresenta o processo de recuperação do código do U-Boot por meio da utilização do GNU/Linux Ubuntu. Na primeira linha é criado um diretório com o comando mkdir, com o nome uboot; depois disso, utiliza-se o comando cd para alterar o diretório corrente para o uboot; com o comando git clone git://git.denx.de/u-boot.git se inicia o processo para recuperar a última versão do código-fonte (LINUX-SUNXI, [s.d.]).

Figura 3.13 | Processo de recuperação do código do U-Boot

```
> mkdir uboot
> cd uboot
> git clone git://git.denx.de/u-boot.git
Cloning into 'u-boot'...
remote: Counting objects: 512760, done.
remote: Compressing objects: 100% (84715/84715), done.
Receiving objects:  20% (103760/512760), 28.72 MiB | 13.00 KiB/s
```

Fonte: elaborada pelo autor.

Após o processo de recuperação do código é necessário fazer a compilação do U-Boot com as configurações específicas para o *target* em específico, considerando configurações de memória, processadores e arquitetura. A Figura 3.14 apresenta as configurações que acompanham o código do U-Boot, certas placas já acompanham configurações para diversos *bootloaders*.

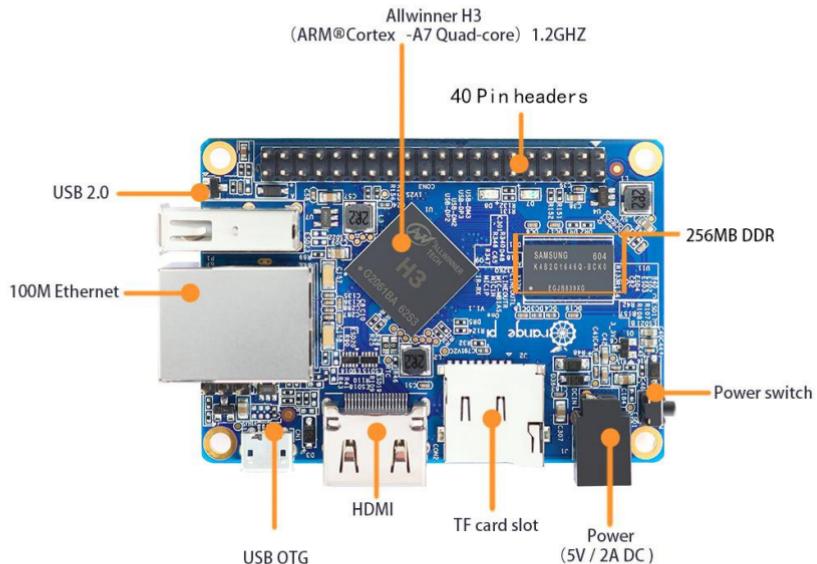



Figura 3.14 | Listagem das configurações das placas

```
ADCIOP.h         cogent_mpc8260.h  DASA_SIM.h     FADS823.h     hymod.h        lubbock.h       NETVIA.h   PN62.h             sbc8260.h     TQM850L.h
ADS860.h         cogent_mpc8xx.h   dnp1110.h      FADS850SAR.h  IAD210.h       lwmon.h         NX823.h    ppmc8260.h         sc520_cdp.h   TQM855L.h
AmigaOneG3SE.h   CPCI4052.h        DU405.h        FADS860T.h    ICU862.h       MBX860T.h       OCRTC.h    R360MPI.h          SCM.h         TQM860L.h
AMX860.h         CPCI405.h         EBONY.h        FLAGADM.h     impa7.h        MBX.h           ORSG.h     RPXClassic.h       shannon.h     trab.h
AR405.h          CPCI440.h         ELPPC.h        FPS850L.h     IP860.h        MHPC.h          OXC.h      RPXlite.h          SM850.h       utx8245.h
BAB7xx.h         CPCIISER4.h       ep7312.h       FPS860L.h     IPHASE4539.h   MIP405.h        PCI405.h   RPXsuper.h         smdk2400.h    W7OLMC.h
BMW.h            CPU86.h           ep8260.h       GEN860T.h     IVML24.h       ML2.h           PCIPPC2.h  RRvision.h         smdk2410.h    W7OLMG.h
c2mon.h          cradle.h          ERIC.h         GENIETV.h     IVMS8.h        MOUSSE.h        PCIPPC6.h  rsdproto.h         SPD823TS.h    WALNUT405.h
CANBT.h          CRAYL1.h          ESTEEM192E.h   GTH.h         KUP4K.h        MPC8260ADS.h    pcu_e.h    sacsng.h           SXNI855T.h    ZUMA.h
CCM.h            csb226.h          ETX094.h       gw8260.h      LANTEC.h       MUSENKI.h       PIP405.h   Sandpoint8240.h    TQM823L.h
cogent_common.h  CU824.h           EVB64260.h     hermes.h      lart.h         MVS1.h          PM826.h    Sandpoint8245.h    TQM8260.h
```

Fonte: elaborada pelo autor.

Neste momento poderíamos iniciar o processo de configuração utilizando o código-fonte original do U-Boot. Todavia, cada placa de sistema embarcado que possui suporte ao U-Boot faz modificações e disponibiliza uma versão em específico deste *bootloader*, como exemplo utilizaremos o Orange Pi One. A Figura 3.15 apresenta uma descrição do Orange Pi One com um processador ARM, 256 DDR e diversos dispositivos de entrada e saída (ISIKDAG, 2015).

Figura 3.15 | Dispositivos do Orange Pi One

Fonte: <http://www.orangepi.org/orangepione/>. Acesso em: 4 dez. 2017.

Para que o U-Boot tenha suporte para o *Orange Pi One* é necessário buscar o código-fonte. A Figura 3.16 mostra como buscar o código-fonte: na primeira linha se cria um diretório *orangepi* e, com o comando *cdi*, torna o *orangepi* o diretório corrente; depois, com o comando *git clone git://git.denx.de/u-boot.git*, busca-se o código-fonte do U-Boot para o Orange Pi.

Figura 3.16 | Recuperando o código-fonte do U-Boot para o *Orange Pi One*

```
> mkdir orangepi
> cd orangepi/
> git clone git://git.denx.de/u-boot.git
Cloning into 'u-boot'...
remote: Counting objects: 512760, done.
remote: Compressing objects: 100% (84715/84715), done.
remote: Total 512760 (delta 422335), reused 510612 (delta 420412)
Receiving objects: 100% (512760/512760), 104.84 MiB | 1.89 MiB/s, done.
Resolving deltas: 100% (422335/422335), done.
> ls
u-boot
> cd u-boot/
> ls
api   board  common     configs  doc            drivers  env       fs       Kbuild   lib       MAINTAINERS  net   README   snapshot.commit  tools
arch  cmd    config.mk  disk     Documentation  dts      examples  include  Kconfig  Licenses  Makefile     post  scripts  test
> █
```

Fonte: elaborada pelo autor.

Para compilar o U-Boot para o Orange Pi One é necessário utilizar os seguintes comandos:

1. CROSS_COMPILE=arm-linux-gnueabi- make orangepi_one_defconfig.

2. CROSS_COMPILE=arm-linux-gnueabi- make.

Nesse processo, o comando *CROSS_COMPILE* informa qual compilador utilizar, que nesse caso é o compilador para a arquitetura ARM (arm-linux-gnueabi-). Na primeira linha é feita a configuração inicial com o *make orangepi_one_defconfig*, e depois apenas com o comando *make* é realizada a compilação. A Figura 3.17 apresenta o resultado do comando inicial CROSS_COMPILE=arm-linux-gnu-make orangepi_one_defconfig, com ele todas as configurações para a compilação são criadas.

Figura 3.17 | Configuração do U-Boot para o Orange Pi one

```
> CROSS_COMPILE=arm-linux-gnueabi- make orangepi_one_defconfig
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
```

Fonte: elaborada pelo autor.

A Figura 3.18 apresenta as últimas mensagens de compilação do Orange Pi. Após a compilação é necessário apenas copiar o *bootloader* para o cartão SD com o comando:

- dd if=u-boot.bin of=/dev/sda bs=1024 seek=8.

O comando *dd* faz a cópia de dados de um dispositivo de origem (if) para o de destino (of). Nesse caso, a origem consiste no resultado da compilação do U-Boot (u-boot.bin) para a primeira partição do cartão SD (/dev/sda).

Figura 3.18 | Final da compilação do U-boot para o Orange Pi

```
  MKSUNXI spl/sunxi-spl.bin
  OBJCOPY u-boot-nodtb.bin
  CAT     u-boot-dtb.bin
  COPY    u-boot.bin
  MKIMAGE u-boot.img
  COPY    u-boot.dtb
  BINMAN  u-boot-sunxi-with-spl.bin
  OBJCOPY u-boot.srec
  SYM     u-boot.sym
  MKIMAGE u-boot-dtb.img
  CHK     include/config.h
  CFG     u-boot.cfg
  CFGCHK  u-boot.cfg
```

Fonte: elaborada pelo autor.

Seguindo a documentação do projeto (LINUX-SUNXI, [s.d.]), após a cópia do *bootloader* no sistema de arquivos no diretório *syslinux* é necessário criar o arquivo chamado *extlinux.conf* com o conteúdo a seguir:

setenv bootargs console=ttyS0,115200 root=/dev/mmcblk0p2 rootwait panic=10

load mmc 0:1 0x43000000 ${fdtfile} || load mmc 0:1 0x43000000 boot/${fdtfile}

load mmc 0:1 0x42000000 uImage || load mmc 0:1 0x42000000 boot/uImage

bootm 0x42000000 – 0x43000000

Esse arquivo define os argumentos de *boot*:

- console=ttyS0,115200: qual é o terminal e a velocidade do terminal.
- root=/dev/mmcblk0p2: partição do cartão em que está o root filesysten.
- rootwait: espera as partições ficarem disponíveis.
- panic: aguarda 10 segundos antes de interromper o processo de boot em caso de erro crítico.

As outras linhas indicam posições de memória específica de cada placa de desenvolvimento. Após todas essas configurações é possível ter um sistema operacional iniciável.

Reflita

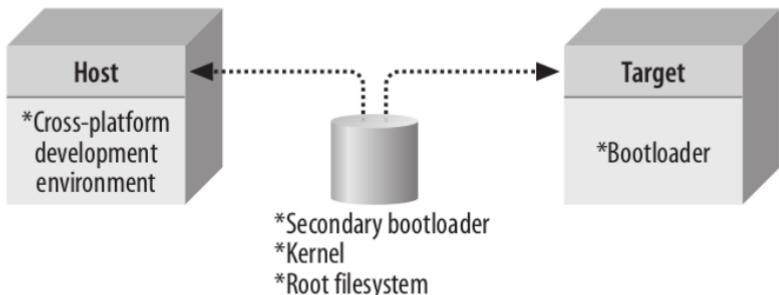
As configurações de um *bootloader* são necessárias para que seja possível utilizar o sistema operacional de um sistema embarcado. Todos os passos são intrincados e qualquer erro pode levar a um sistema que não inicie corretamente. Em um sistema embarcado, sem sistema operacional, o *bootloader* deve ser tão complexo?

Sem medo de errar

O grande projeto da empresa KtechPi consiste na construção de diversos sistemas embarcados para monitorar a cidade, seguindo os conceitos de uma smartcity (cidade equipada com diversos

sensores e atuadores para monitorar e controlar semáforos, sistema contra inundação e outros), em que aspectos de temperatura, umidade, intensidade do sol, qualidade do ar e outros elementos são monitorados. Assim, diversos sistemas embarcados foram instalados pela cidade já em seu local final, com rede de fibra óptica.

Em primeiro lugar, será monitorada a qualidade do ar com sensor experimental. O software que faz a leitura dos dados sofrerá diversas mudanças no processo de desenvolvimento, objetivando melhorias. Todavia, por razões contratuais, esses equipamentos já serão instalados em seu local final com comunicação por uma rede de fibra óptica.

Para demonstrar a solução para esse projeto, você deveria fazer uma apresentação contendo:

1. Os conceitos do processo de inicialização de um sistema embarcado.
2. O tipo de arquitetura utilizada.
3. Qual *bootloader* utilizar.

Assim, os principais conceitos que devem aparecer na apresentação são:

- O *bootloader* é o elemento que faz as configurações inicias de um sistema embarco. Sua principal função é indicar onde está o kernel e o sistema de arquivo principal.
- O *bootloader* deve ser relacionado à arquitetura de hardware e ao modelo de utilização do sistema.
- O sistema embarcado pode receber configurações como rede e vídeo no *bootloader*.

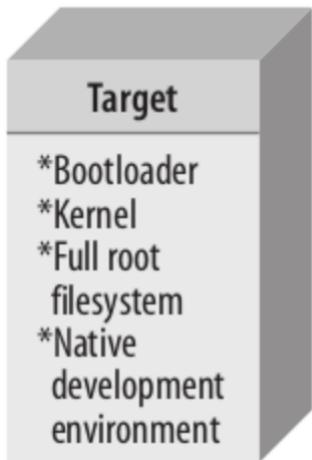
O tipo de arquitetura a ser utilizada seria a com o *host* e *target*, com armazenamento remoto do *kernel* e sistemas de arquivos conforme o exemplificado na figura a seguir, pois é possível alterar o *firmware* de todos os sistemas da cidade sem ser necessário acessar o local onde os equipamentos estão instalados.

Figura 3.19 | Sistema de boot com imagem remota

Fonte: Yaghmour (2003, p. 40)

O *bootloader* indicado depende da arquitetura, nesse caso, é interessante utilizar um sistema simples e pequeno para que seja espalhado na cidade. É recomendável utilizar o U-Boot para que na inicialização do sistema ao invés de buscar a imagem do sistema no disco local, o sistema embarcado, utilizando as configurações do U-Boot, possa buscar uma imagem atualizada na rede.

Avançando na prática

Sistema de medição de gases perigosos

Descrição da situação-problema

A empresa MonitorCom está testando um sistema embarcado com sensores para gases perigosos, como o arsênico. Para esses testes é necessário colocar uma placa do Orange Pi dentro de uma caixa de acrílico selada e liberar quantidades do gás tóxico para fazer calibração, testes e desenvolvimento do sistema. Dessa forma, é necessário indicar qual arquitetura de sistema embarcado será utilizada e qual a forma que o *bootloader* vai atuar para que seja possível fazer todo o desenvolvimento sem que sejam necessárias modificações no hardware para atualizar o *firmware*.

Resolução da situação-problema

Para esse cenário será utilizada uma arquitetura que contenha apenas o *target*, como mostra a Figura 3.20.

Figura 3.20 | Arquitetura de um sistema embarcado baseado apenas no *target*

Fonte: Yaghmour (2003, p. 41)

O *bootloader* terá o papel apenas de apontar para o armazenamento local do sistema embarcado onde está o *kernel* e o sistema de arquivos. Com isso, a imagem que está escrita no sistema embarcado não será alterada e todas as ferramentas de desenvolvimento já estarão no sistema embarcado. Assim, são minimizadas as chances de ter de abrir a caixa com o sistema embarcado e ainda é possível continuar o desenvolvimento dele.

Faça valer a pena

1. Os sistemas embarcados podem ter diversas arquiteturas para seu desenvolvimento, cada uma delas possui elementos específicos que podem ser facilitadores de certos cenários. A arquitetura apenas com o *target* deve conter todos os elementos de desenvolvimento.

Em uma arquitetura com apenas o *target*, quais são os elementos do sistema embarcado que devem constar?

a) Bootloader, kernel, full root file system e native development environment.
b) Bootloader.
c) Root file system e native development environment.
d) NFS e bootloader.
e) Bootloader, full root file system e native development environment.

2. O *bootloader* permite que o sistema embarcado tenha sua primeira configuração e seja utilizável. Para um sistema baseado em computadores da arquitetura X86, podemos utilizar o GRand Unified Bootloader (GRUB) e, no caso de *Advanced RISC Machine* (ARM), o U-Boot.

Os *bootloader* podem ter diversas funções para prover as configurações de um sistema embarcado, como principal temos:

a) Configuração do *timeout* de rede.
b) Configuração do sistema de arquivos.
c) Configuração do *kernel*.
d) Indicação de onde está o kernel e o sistema de arquivos.
e) Configuração da rede.

3. Um sistema embarcado pode ter sua inicialização feita de diversas formas, buscando os arquivos localmente ou ainda fazendo acesso pela rede. Quando se busca os dados pela rede é necessário ter em mente que o *bootloader* deverá carregar os módulos e configurar a interface de rede para que seja possível a utilização do sistema embarcado.

Existem diversos serviços que um sistema embarcado pode utilizar para sua operação, todavia, existe um de grande valia para que ele busque o *kernel* e o sistema de arquivos de forma remota. Assinale a alternativa correta.

a) NFS.
b) TCP.
c) TFTP.
d) UDP.
e) ICMTP.

Seção 3.3

Configuração de redes de sistemas embarcados

Diálogo aberto

A agricultura está cada vez mais precisa e dependente da tecnologia; o processo produtivo está sendo acompanhado de perto para que se maximize os lucros e diminua os custos. Todavia, as questões de saúde relacionadas às novas descobertas da ciência no campo da agricultura têm entrado em pauta nas discussões sobre esse tema. Nesse sentido, é muito necessário o monitoramento de plantações para descobrir o melhor momento de aplicação dos agrotóxicos ou outros procedimentos para controle de pragas.

Considerando essa preocupação do uso consciente do agrotóxico, a empresa AgroView tem um projeto que utilizará um sistema embarcado para monitorar lavouras. O projeto consiste em utilizar uma placa, como um Raspberry Pi ou Orange Pi, acoplado a uma câmera e a um software embarcado que processará as imagens in loco e, em caso da detecção de algum tipo de praga, o sistema enviará um alerta via SMS e será possível acessar o sistema para ver as imagens por um navegador web.

Você, sendo parte da equipe de desenvolvimento, tem de apresentar para a gerência da empresa as vantagens de utilizar um sistema mais geral, como o Raspberry Pi, em comparação ao sistema com hardware mais restrito. Para isso, você deverá produzir uma apresentação de três slides para ser feita em 5 minutos (apresentação do tipo pitch). Essa apresentação precisa conter a comparação da instalação do serviço de servidor de página para uma placa de uso mais geral, como Raspberry Pi, e um hardware de uso dedicado.

Não pode faltar

O desenvolvimento de um sistema embarcado pode ser visto em três grandes etapas. A primeira consiste em escolher e preparar o hardware; a segunda etapa prepara e configura o sistema operacional básico; por fim, na terceira etapa, preparam-se todos

os serviços e softwares que utilizarão o hardware e o sistema operacional (OLIVEIRA, 2010). Como foi visto em seções anteriores, para que seja possível o desenvolvimento do sistema, passaremos pelas seguintes etapas descritas no Quadro 3.4.

Quadro 3.4 | Etapas para o desenvolvimento de sistemas embarcados.

	Etapa	Descrição
1	Criação do *root filesystem*.	Sistema que controla como os dados são armazenados em memória não volátil dos sistemas embarcados.
2	Configuração e compilação das ferramentas do sistema operacional.	Criação de todos os comandos e funções para utilizar o *hardware*.
3	Configuração e compilação do *kernel*	Configurar o *kernel*, que tem por função primordial acessar o *hardware*.
4	Formação do sistema de arquivo	Define como será arquitetada a memória de armazenamento do sistema embarcado.
5	Instalação do *bootloader*	*Software* que faz a inicialização e configuração inicial do sistema de arquivos e inicializa o *kernel*.
6	**Instalação da aplicação foco do sistema embarcado**	**Insere no *firmware* do sistema embarcado as funções que darão a utilização ao sistema embarcado.**
7	Envio do *firmware* para o sistema *target*	O *firmware* é o software enviado para a placa que executará o sistema embarcado (*target*).

Fonte: Yaghmour (2003) e Sally (2010).

Neste momento, estamos na etapa de número 6, instalação da aplicação foco do sistema embarcado. A aplicação ou software de que dará a função do sistema embarcado tem relação direta com os requisitos da produção do equipamento. Por exemplo, um sistema que fará o monitoramento dos batimentos cardíacos receberá um software que faz a leitura do sensor do coração, mas para que isto seja possível são necessários alguns serviços do sistema operacional. Esses serviços podem estar relacionados a configuração de rede, acesso remoto, monitoramento e outros. Alguns exemplos de serviços (YAGHMOUR, 2003):

- *Simple Network Management Protocol* (SNMP): protocolo que provê diretrizes para monitorar o sistema embarcado.
- *Secure Shell* (SSH): serviços de acesso remoto.
- *Servidor Hypertext Transfer Protocol* (HTTP): serviço que provê acesso via HTTP para páginas web.
- *Firewall*: sistema para filtragem de pacotes de rede.

A instalação dessas aplicações em sistema GNU/Linux para computadores em geral é feita com as ferramentas de pacotes de cada aplicação. Todavia, hoje existem placas para desenvolvimento de aplicações embarcadas que propiciam a fácil implementação de um serviço, sem a necessidade de passar por todas as etapas descritas no Quadro 3.4. Como exemplo, temos o Raspberry Pi 3, apresentado na Figura 3.21, que segundo Isikdag (2015) possui um processador Advanced RISC Machine (ARM), 802.11n *Wireless LAN*, 4 portas USB, entrada para cartão micro Secure Digital (SD) e diversos outros elementos. Para instalar o *Raspbian* (sistema operacional para o Raspberry baseado a na distribuição GNU/Linux Debian) é necessário um cartão MicroSD e um computador de uso geral com interface para MicroSD. Utilizando um GNU/Linux como o Debian ou Ubuntu, após fazer o download da imagem, é preciso descompactá-la e gravá-la no cartão SD. A Figura 3.22 apresenta o processo de download com o comando *wget*; com o comando *unzip* acontece a descompactação; e, por fim, no comando *dd* acontece o processo de escrita no cartão.

Figura 3.21 | Visão do Raspberry Pi 3

Fonte: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. Acessado em: 10 dez. 2017.

O processo de criação de um sistema embarcado é complexo e requer planejamento. Todavia, hoje é possível utilizar placas para desenvolvimento, como o *Raspberry pi* ou *Orange pi*, que já constam com diversos recursos de hardware e software. Porém, os custos computacionais (memória, processador, etc.) são maiores que uma plataforma desenvolvida de forma específica ou, ainda, do que placas que necessitam que seja produzido seus softwares com maiores limitações computacionais. Qual seria o *trade-off* (equilíbrio para considerar duas ou mais situações) para escolher uma plataforma já pronta ou fazer o seu desenvolvimento?

Figura 3.22 | Processo de download, descompactação e escrita no cartão SD

```
> wget http://ftp.jaist.ac.jp/pub/raspberrypi/raspbian/images/raspbian-2017-12-01/2017-11-29-raspbian-stretch.zip
--2018-01-02 13:39:12-- http://ftp.jaist.ac.jp/pub/raspberrypi/raspbian/images/raspbian-2017-12-01/2017-11-29-raspbian-stretch.zip
Resolving ftp.jaist.ac.jp (ftp.jaist.ac.jp)... 150.65.7.130, 2001:df0:2ed:feed::feed
Connecting to ftp.jaist.ac.jp (ftp.jaist.ac.jp)|150.65.7.130|:80... connected.
HTTP request sent, awaiting response... 200 OK
```

```
Length: 1764972666 (1.6G) [application/zip]
Saving to: '2017-11-29-raspbian-stretch.zip.2'

2017-11-29-raspbian-stretch.zip
100%[==================================================>] 1.64G   1.15MB/s in 29m 44s

> unzip 2017-11-29-raspbian-stretch.zip
Archive: 2017-11-29-raspbian-stretch.zip
  inflating: 2017-11-29-raspbian-stretch.img
> dd bs=4M if=2017-11-29-raspbian-stretch.img of=/dev/sda conv=fsync
```

Fonte: elaborada pelo autor.

Neste tipo de sistema operacional embarcado, baseado em uma distribuição GNU/Linux completa, o processo de instalação é simplificado, pois utiliza o gerenciador de pacotes da distribuição; no caso do *Raspbian* o comando *apt-get* faz o processo de instalação. A Figura 3.23 apresenta o processo de instalação do servidor de páginas Apache.

Figura 3.23 | Instalação do servidor HTTP Apache

```
> apt-get install apache2
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  linux-image-3.16.0-4-amd64
Use 'apt autoremove' to remove it.
The following additional packages will be installed:
  apache2-data apache2-utils
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom
The following NEW packages will be installed:
  apache2 apache2-data apache2-utils
0 upgraded, 3 newly installed, 0 to remove and 60 not upgraded.
Need to get 614 kB of archives.
After this operation, 1,863 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://ftp.br.debian.org/debian stretch/main amd64 apache2-utils amd64 2.4.25-3+deb9u3 [217 kB]
Get:2 http://ftp.br.debian.org/debian stretch/main amd64 apache2-data all 2.4.25-3+deb9u3 [162 kB]
Get:3 http://ftp.br.debian.org/debian stretch/main amd64 apache2 amd64 2.4.25-3+deb9u3 [235 kB]
Fetched 614 kB in 0s (1,503 kB/s)
```

Fonte: elaborada pelo autor.

Todavia, o *Raspberry Pi* 3 tem como requisito mínimo um cartão SD de 8 GB, o que representa um grande volume de dados quando se projeta um *hardware* modelado para um requisito específico. Dessa forma, em um processo para um *hardware* embarcado delineado são necessárias formas de utilizar menos memória para as instalações. Essa escolha deve ser feita dados os requisitos de projeto, considerando limitações de preço, orçamento, tempo de desenvolvimento e outros (GANSSLE, 2008).

Seguindo os requisitos e as limitações do hardware não é indicado instalar um servidor de páginas como o Apache pelos seus diversos requisitos de bibliotecas externas e tamanho de instalação final. Dessa forma, uma alternativa é o *thttpd*, que fornece diversas funcionalidades semelhantes às do Apache e possui tamanho final de aproximadamente 72 KB, sendo que o Apache possui cerca de 6 MB em sua instalação básica. Para fazer a instalação do *thttpd* (<https://acme.com/software/thttpd/> acesso em: 9 mar. 2018) deve-se fazer a compilação para a arquitetura-alvo, lembrando a estrutura básica de organização. No *host* é necessário criar a seguinte estrutura de diretórios: *nomeDoSistemaEmbarcado*.

- *src*.
- *build*.
- *target*.

No diretório *src* ficarão todos os códigos-fonte dos elementos de software que serão utilizados no projeto, nesse caso, faremos o download e inseriremos na pasta *src*. Na pasta *build* ficarão todos os arquivos derivados do processo de compilação, e na pasta *target* consta toda estrutura que será enviada para o sistema *target*. Para o processo de compilação devemos utilizar o comando *cd* para ir para o diretório *src* e criar um diretório chamado *thttpd*, com o comando *mkdir* e novamente o comando cd para ir ao diretório recém-criado.

Para a configuração da interface de rede é necessário atentar aos arquivos de configuração de cada disco ou aos arquivos que fazem parte da inicialização do sistema. A configuração mais geral é feita com o comando *ifconfig* e *route* (SALLY, 2010):

- ifconfig eth0 192.168.0.5 netmask 255.255.0.0
- route add default gw 192.168.0.5

O comando *ifconfig* tem como parâmetro a interface em que será realizada a configuração (eth0 é primeira interface de rede), 192.168.0.5 é o endereço de Internet Protocol (IP) e, por fim, o *netmask* é a máscara de rede. O outro comando consiste em definir qual é o IP que o sistema deve consultar caso a requisição não seja da rede local; o comando *route* tem como parâmetro o *add* que informa para acionar uma rota padrão (*default gw*) com o IP indicado.

Existem diversas alternativas de software para serviços de rede para sistemas embarcados. Essas outras opções sempre focam em tamanhos de binários menores comparados às soluções clássicas. Por exemplo, para serviço de *SSH* pode ser utilizado o Dropbear (<http://matt.ucc.asn.au/dropbear/dropbear.html>. Acesso em: 9 mar. 2018) ou, para o Apache, o Boa (<http://www.boa.org/>. Acesso em: 9 mar. 2018).

Os próximos passos consistem em fazer o download do código-fonte com o comando *wget*, descompactar com o comando *tar* e, por fim, executar a compilação com os comandos configure e *make*. Na Figura 3.24 é possível ver o processo de criação das pastas, o download do código-fonte e a descompactação.

Figura 3.24 | Criação da estrutura de diretórios e download do código-fonte do *thttpd*

```
> cd src/
> mkdir thttpd
> cd thttpd
> wget https://fossies.org/linux/www/thttpd-2.27.tar.gz
–2018-01-02 13:45:20–  https://fossies.org/linux/www/thttpd-2.27.tar.gz
Resolving fossies.org (fossies.org)... 138.201.17.217
Connecting to fossies.org (fossies.org)|138.201.17.217|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 134005 (131K) [application/x-gzip]
Saving to: 'thttpd-2.27.tar.gz'

thttpd-2.27.tar.gz              100%[================================================>]
130.86K   162KB/s    in 0.8s

2018-01-02 13:45:22 (162 KB/s) - 'thttpd-2.27.tar.gz' saved [134005/134005]

> tar -xf thttpd-2.27.tar.gz
```

Fonte: elaborada pelo autor.

O próximo passo consiste em fazer a compilação do código-fonte para a arquitetura do *target*. Para isso é necessário utilizar o comando: CC=arm-linux-gnueabi-gcc./configure –host=/home/projeto/*nomeDoSistemaEmbarcado/target*. A opção CC indica qual compilador utilizar, nesse caso, será o compilador para ARM *arm-linux-gnueabi-gcc*, o comando configura e prepara todos os arquivos com as opções de compilador e nome do computador *target* com a opção *host*. Após esse processo é necessário utilizar o comando *make* para inciar a compilação. A Figura 3.25 apresenta o final do processo de compilação do *thttpd*, mesmo com alguns avisos é possível reparar que não ocorreu nenhum erro no processo.

Assimile

O processo de compilação para aplicações para sistemas embarcados sempre é direcionado para a arquitetura do *target*. O compilador deve gerar binários que possam ser executados no processador de *target*.

Figura 3.25 | Mensagens finais do processo de compilação do *thttpd*

```
> make
arm-linux-gnueabi-gcc -O2 -DHAVE__PROGNAME=1 -DHAVE_FCNTL_H=1 -DHAVE_GRP_H=1 -
DHAVE_MEMORY_H=1 -DHAVE_PATHS_H=1 -DHAVE_POLL_H=1 -DHAVE_SYS_POLL_H=1 -
DTIME_WITH_SYS_TIME=1 -DHAVE_DIRENT_H=1 -DHAVE_LIBCRYPT=1 -DHAVE_STRERROR=1
-DHAVE_WAITPID=1 -DHAVE_VSNPRINTF=1 -DHAVE_DAEMON=1 -DHAVE_SETSID=1 -
DHAVE_GETADDRINFO=1 -DHAVE_GETNAMEINFO=1 -DHAVE_GAI_STRERROR=1 -
DHAVE_SIGSET=1 -DHAVE_ATOLL=1 -DHAVE_UNISTD_H=1 -DHAVE_GETPAGESIZE=1 -
DHAVE_SELECT=1 -DHAVE_POLL=1 -DHAVE_TM_GMTOFF=1 -DHAVE_INT64T=1 -
DHAVE_SOCKLENT=1 -I. -c thttpd.c
thttpd.c: In function 'main':
thttpd.c:611:12: warning: implicit declaration of function 'sigset' [-Wimplicit-function-declaration]
    (void) sigset( SIGTERM, handle_term );
           ^~~~~~
```

Fonte: elaborada pelo autor.

Com o processo de compilação realizado, é necessário copiar o binário para a pasta *target* com o comando *cp thttpd ../../../target/thttpd*. Por fim, é preciso remover as ligações que os binários possuem com o sistema *host*, para que seja possível utilizar o binário no *target* com o comando: arm-linux-gnueabi-strip ../../../*target/thttpd*. Para finalizar o processo é necessário fazer a configuração do

servidor criando o arquivo *thttpd.conf*, que deve ficar na pasta /etc do *target*, no Quadro 3.5 temos um exemplo do conteúdo padrão do arquivo. Na opção *dir*, o arquivo apresenta o diretório onde ficarão os arquivos disponibilizados, a opção *logfile* informa onde ficarão os *logs* de acesso e a opção *pidfile* apresenta o arquivo que informa qual é o identificador do processo que está controlando o *thttpd*.

Quadro 3.5 | Arquivo de configuração do *thttpd*

```
# This section overrides defaults
dir=/home/httpd/html
logfile=/var/log/thttpd.log
pidfile=/var/run/thttpd.pid
```

Fonte: elaborado pelo autor.

O processo de instalação dos serviços para rede em sistemas embarcados com hardware dedicado sempre é baseado na compilação e configuração manual dos pacotes. Mesmo sendo uma etapa mais complexa é necessária para evitar uso demasiado de recursos.

Em sistemas baseados em GNU/Linux as ferramentas de desenvolvimento são instaladas pelo gerenciamento de pacotes da distribuição. Para o sistema Microsoft Windows é necessário instalar o Visual Studio IoT (BELL, 2016) e os pacotes são em sua grande maioria diretamente relacionados a Microsoft, veja alguns:

- <https://www.youtube.com/watch?v=_tCANJXmPDw>. Acesso em: 9 mar. 2018.
- <https://developer.microsoft.com/en-us/windows/iot/samples/helloblinkyserver>. Acesso em: 9 mar. 2018.
- BORYCKI, Dawid. **Programming for the internet of things**: using Windows 10 IoT core and azure IoT suíte. [S.l.]: Microsoft Press, 2017.
- ALMEIDA, Rodrigo Maximiano Antunes de; MORAES, Carlos Henrique Valério de; SERAPHIM, Thatyana de Faria Piola. **Programação de sistemas embarcados**. [S.l.]: Elsevier, 2017.
- <https://www.youtube.com/watch?v=vAK2LFtpbhI>. Acesso em: 2 jan. 2018.
- <https://youtu.be/2auIrISGpag>. Acesso em: 2 jan. 2018.

Sem medo de errar

A empresa AgroView tem um novo projeto para monitoramento de plantações baseado em sistema embarcado. Haverá uma câmera, cujas imagens serão projetadas pelo sistema, caso algum problema for detectado as imagens serão disponibilizadas para o servidor de páginas web, para a análise dos técnicos. Com base nessas informações, a empresa poderá tomar a decisão de aplicar agrotóxico ou fazer outras ações de controle.

Seu objetivo é comparar a instalação de um sistema de página web em um hardware embarcado de uso geral, como o *Raspberry PI* ou outro hardware que será projetado e delineado com limitações de recursos.

Para isso, você precisará preparar uma apresentação com três slides, para ser feita em 5 minutos (do tipo *pitch*). Essa apresentação deve conter a comparação da instalação do serviço de servidor página para uma placa de uso mais geral, como *Raspberry Pi*, e um hardware de uso dedicado. Essa apresentação deve conter os elementos comparativos, como vemos no Quadro 3.6. O primeiro elemento de comparação são as formas de instalação dos elementos de sistema operacional e software, o segundo é o tempo utilizado para o desenvolvimento de cada tipo de arquitetura e o último é a forma de depuração necessária em cada cenário.

Quadro 3.6 | Comparação entre hardware embarcado de uso geral e de uso específico

Característica	Hardware para sistema embarcado de uso geral	Hardware para sistema embarcado específico
Compilação/instalação	O processo é feito com as ferramentas do sistema operacional disponibilizado pelo fabricante.	É necessária a compilação de todas as etapas.
Tempo de desenvolvimento	Como o projeto já está pronto é possível se dedicar mais ao software.	No hardware específico é necessário desenvolver o hardware e o software.
Depuração	Como existem diversas interfaces no hardware é possível utilizar recursos como vídeo ou outros elementos para *debug*.	A interface de *debug* deve ser construída no projeto

Fonte: elaborado pelo autor.

Avançando na prática

Sistema de segurança para monitoramento de áudio

Descrição da situação-problema

Um sistema que grava e processa o áudio em um ambiente utiliza, em sua construção, um conjunto de bibliotecas de processamento de áudio e reconhecimento de fala. Caso detecte alguma "frase-chave" ele aciona os alarmes para a equipe de segurança. Além disso, o sistema deve possuir um serviço de rede equipado com o *Simple Network Management Protocol* (SNMP), para que caso o sistema pare de funcionar a equipe de segurança também seja avisada. Todavia, a equipe está com dúvidas sobre a melhor forma de fazer o processo em um sistema baseado em *Raspberry Pi*, se deve fazer a instalação pelo sistema de gerenciamento de pacotes ou a compilação dos pacotes para o serviço de SNMP. Faça um conjunto de passos para instalar via compilação e via o gerenciador de pacotes de uma distribuição como o *Raspberry Pi*.

Resolução da situação-problema

Essa tarefa precisa da utilização de um servidor SNMP, para isso, pode ser feita a compilação. Em um sistema baseado no *Raspberry Pi* é possível fazer a instalação seguindo os passos:

1. Atualizar a base de pacotes - *apt-get upgrade*
2. Instalar o sevidor de SNMP - *apt-get install snmpd snmp*

Para que o servidor seja compatível com a arquitetura do sistema alvo, é preciso fazer a instalação utilizando o código-fonte:

1. Fazer o download de um servidor de SNMP, como o Net-SNMP: utilizando o comando *wget* em <http://www.net-snmp.org/download.html>. Acesso em: 9 mar. 2018.
2. Descompactar o arquivo: *tar -xf net-snmp-5.7.3.tar.*
3. Configurar a compilação: *./configure.*
4. Compilar: *make.*
5. Instalar: *make install.*

Faça valer a pena

1. Em sistemas embarcados com sistema operacional que utiliza um hardware dedicado é necessário compilar ou buscar versões dos softwares para a arquitetura do sistema target. Todavia, esse processo traz mais complexidade ao projeto e, por sua vez, gera um tempo maior de desenvolvimento.

O processo de compilação de um software para um sistema embarcado no ambiente GNU/Linux deve passar pelos processos:

a) Configuração, compilação/linker e instalação.
b) Configuração e instalação.
c) Configuração, compilação e instalação.
d) Apenas compilação.
e) Apenas linker.

2. Os serviços de rede de um sistema embarcado têm um papel importante, sendo que vários deles fazem parte da aplicação final do sistema. Todavia, em diversos casos, não é possível utilizar os softwares clássicos para cada serviço em um sistema embarcado. Para SSH, por exemplo, pode-se utilizar o servidor Drop bear.

Por que o uso de softwares clássicos para os serviços de rede em sistemas embarcados não é indicado ou não é possível?

a) Por causa do tamanho final do binário e do tempo de compilação.
b) Por causa do tamanho final e da dependência de bibliotecas.
c) Por causa do tempo de compilação.
d) Por causa da utilização de hardware dedicado.
e) Porque o compilador não confiável.

3. Para a utilização de um serviço de rede em sistema embarcado é necessário configurar a rede para seja possível a comunicação. O processo de configuração tem duas etapas para que o dispositivo possa se comunicar na rede local e para fora, por exemplo, na internet.

Quais são os dois comandos básicos para que seja possível a configuração de rede em sistema embarcado baseado em GNU/Linux?

a) df e free.
b) mkfs.
c) ifconfig e route.
d) route.
e) cd e ls.

Referências

BELL, C. **Windows 10 for the internet of things.** [S.l.]: Apress, 2016.

BERGER, Arnold S. **Hardware and computer organization.** [S.l.]: Elsevier, 2005.

GANSSLE, Jack G.; BARR, Michael. **Embedded systems dictionary.** [S.l.]: Taylor, 2003.

GANSSLE, Jack. **The art of designing embedded systems.** [S.l.]: Elsevier Science, 2008.

ISIKDAG, Umit. **Enhanced building information models using IoT services and integration patterns.** [S.l.]: Springer, 2015.

LINUX-SUNXI. Linux-sunxi. [s.d.]. Disponível em: <http://linux-sunxi.org/Main_Page>. Acesso em: 4 dez. 2017.

NEGUS, Christopher. **Linux Bible 2010 Edition**: Boot Up to Ubuntu, Fedora, KNOPPIX, Debian, openSUSE, and 13 Other Distributions. [S.l.]: Wiley, 2010.

OLIVEIRA, André de; ANDRADE, Fernando de. **Sistemas embarcados**: hardware e firmware na prática. 2. ed. [S.l.]: Érica, 2010.

ORANGEPI. ORANGEPI Documentation. [s.d.]. Disponível em: <http://www.orangepi.org/Docs/Building.html>. Acesso em: 4 dez. 2017.

PAVLOV, Stanislav; BELEVSKY, Pavel. **Windows embedded CE 6.0 fundamentals**. 4. ed. [S.l.]: Microsoft Press, 2008.

SALLY, Gene. **Pro Linux embedded systems**. [S.l.]: Apress, 2010.

STALLINGS, William. **Arquitetura e organização de computadores**. 8. ed. [S.l.]: Pearson, 2009.

TANENBAUM, Andrew S. **Organização estruturada de computadores**. [S.l.]: Pearson, 2013.

______. **Sistemas operacionais modernos**. [S.l.]: Pearson, 2008.

YAGHMOUR, Karim. **Building embedded linux systems**. [S.l.]: O'Reilly Media, Inc., 2003.

______. **Embedded android**: porting, extending, and customizing. [S.l.]: O'Reilly Media, Inc., 2013.

ZHANGJIN, Wu; CAO, Ziqiang. **Instant optimizing embedded systems using Busybox.** [S.l.]: ProQuest, 2013.

Sistemas de tempo real, sensores e atuadores

Convite ao estudo

Prezados alunos, essa etapa dos estudos tem o objetivo de afinar as habilidades sobre sistemas embarcados. O primeiro elemento de estudo são as técnicas de depuração de um sistema, a possibilidade de analisar a execução de um sistema é de grande importância para que o produto final tenha a menor quantidade de bugs e ainda em caso de alguma falha seja possível descobrir a origem de forma rápida. Para esse estudo serão abordados um conjunto de temas que são essenciais para a resolução de problemas:

- Ferramentas de depuração de sistemas embarcados.
- Execução de depuração de sistemas embarcados.
- Métodos científicos de sistemas embarcados.
- Técnicas para resolução de problemas de sistemas embarcados.

A segunda etapa apresenta os elementos de um sistema de tempo real, tais sistemas têm por característica responder de forma pronta as requisições da aplicação do sistema embarcado e tais sistemas de tempo real possuem aplicações específicas. O estudo de sistemas em tempo real é essencial para a aplicação correta em diversos cenários, para isso é necessário abordar:

- Introdução aos sistemas de tempo real (RTOS).
- Gerenciamento de tarefas de sistemas embarcados.
- Aspectos funcionais de um RTOS.

- Aplicações de um RTOS.

E por fim, a visão dos atuadores e sensores que podem ser aplicados em um sistema embarcado; tais elementos são de importância para que o produto final possa interagir com o mundo externo. Para que os sensores e atuadores sejam aplicados de forma correta em sistema é importante estudar:

- Sensores de sistemas embarcados.
- Atuadores de sistemas embarcados.
- Interface com usuário para atuadores e sensores de sistemas embarcados.
- Aplicações de sensores e atuadores de sistemas embarcados.

Seção 4.1

Ferramentas de depuração de sistemas embarcados

Diálogo aberto

Durante esta seção o foco são as ferramentas para encontrar defeitos em um código que podem gerar uma falha, esse processo é semelhante nas luzes que indicam algum problema em um carro. Em alguns casos apenas a luzes de indicação podem não ser suficientes, sendo necessário algum tipo de equipamento especial, como os scanners de injeção eletrônica. Em nosso cenário, temos momentos que simples sinais já são suficientes (escrita simples em um arquivo de log) para detectar um problema, em outros casos é necessário ferramentas mais sofisticadas.

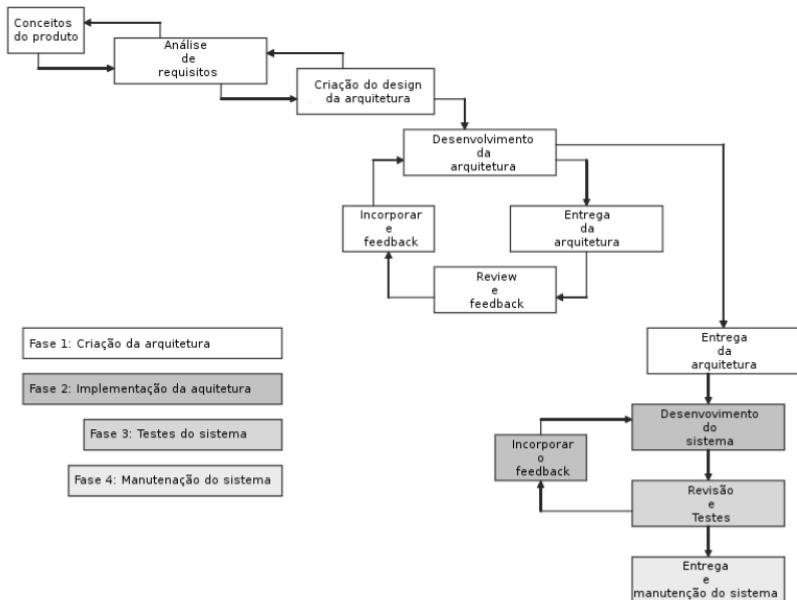
Uma empresa de inovação está com um projeto para a produção de monitoramento sonoro para cidades. O sistema consiste em uma placa Orange PI equipada com um microfone, o software faz a leitura do áudio e analisa buscando mudanças abruptas de áudio e com técnicas de processamento de áudio e inteligência artificial.

Todavia, depois da produção da primeira versão beta (versão em desenvolvimento feita para testes) o sistema está apresentando alguns problemas relatados de dois engenheiros da equipe de testes:

1. Ao deixar o sistema executando por cerca de 4 horas o software simplesmente para de funcionar e para restaurar é necessário retirar a fonte de energia elétrica e ligá-lo novamente.
2. O sistema está em operação regular durante grandes períodos de tempo. Todavia, quando o sistema detecta mais de um evento por minuto seu desempenho decresce rapidamente se tornando indisponível.

A equipe de desenvolvimento precisa de uma direção de como fazer o processo de depuração dos erros reportados pela equipe de testes. Com isso, descreva quais seriam as formas de depurar a aplicação em cada cenário apresentado e um guia passo a passo de como executar o processo.

Não pode faltar

Na unidade anterior os estudos foram baseados no desenvolvimento das estruturas básicas de diretório, no software que configura e cria a inicialização do sistema operacional e os serviços básicos de rede. Com isso, somando os requisitos do sistema embarcado é possível desenvolver uma aplicação para utilizar esse hardware com sistema operacional. Todavia, é importante termos formas de detectar os defeitos no código e evitar as falhas.

No projeto de desenvolvimento de qualquer produto existe sempre a necessidade de executar testes para verificar funcionalidades e requisitos, descrever quais são os ambientes que o sistema pode funcionar e ainda situações onde existem erros na aplicação, sendo necessário fazer a identificação e correção (GANSSLE, 2008). No caso de sistema embarcado temos diversas características peculiares, tais como (OLIVEIRA; ANDRADE, 2010):

- Limitações de interface e hardware.
- Ambiente de aplicação diverso.
- Sistema operacional com ferramentas reduzidas.

A limitação de interfaces de comunicação e de hardware junto com as limitações que o sistema operacional embarcado apresenta são os dois principais elementos que geram mais complexidade em um projeto. Em comparação a um sistema operacional para um computador de uso geral, sempre temos por padrão monitor, teclado ou uma interface de rede para fazer a comunicação. Porém, em um sistema embarcado é necessário planejar e adicionar (diversas vezes com um custo maior) esse tipo de interface. E ainda, dependendo onde esse sistema será testado ou aplicado o seu acesso físico é restrito. Na Figura 4.1, segundo Noergaard (2012) na Fase 1 se avalia os conceitos do produto e deve-se ter em mente quais seriam as necessidades de depuração da aplicação, com base nisso é feita a análise de requisitos (necessidades que são avaliadas dados os conceitos), na terceira etapa é feita a criação do design da arquitetura do projeto considerando como fazer a depuração e com esse projeto é feito o desenvolvimento, testes e feedbacks desse processo utilizando essa interface de debug. No final dessa etapa é criada a arquitetura de final (entrega da arquitetura) e com isso se inicia o desenvolvimento do sistema (fase 2), testado e validado até a entrega

final (fase 3 e 4). Mesmo na entrega final a interface para depuração deve existir, pois técnicos devem ter a possibilidade de acessar o dispositivo ou ainda ser possível dar suporte de forma remota.

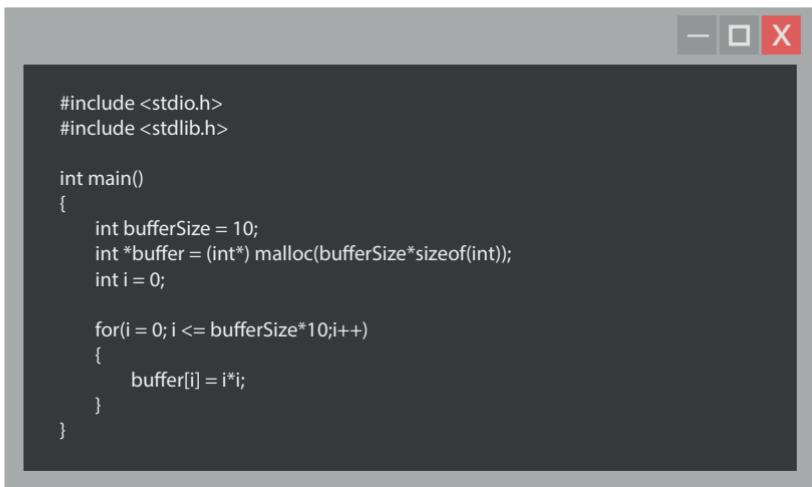
Figura 4.1 | Processo de desenvolvimento de um sistema embarcado

Fonte: adaptada de Noergaard (2012, p. 8).

O processo de detecção de um erro pode ser feito de 3 maneiras (SALLY, 2010):

1. Interativo: o processo de debug interativo consiste em utilizar uma ferramenta auxiliar que consegue marcar certas linhas ou porções do código ou ainda fazer a execução em passos. Nesse cenário é possível analisar linha por linha de código.
2. Utilizando registro de memória: esse tipo de debug pode ser feito quando um problema no software do sistema embarcado ocorre e o conteúdo da memória utilizada pelo software é armazenada na memória não volátil. Com isso, é possível utilizar esse "*dump*" de memória para verificar o conteúdo da memória e encontrar o defeito que gerou a falha.
3. Instrumentação: nesse processo o código do problema é feito equipado com diversos elementos que enviam via rede, terminal, porta serial ou outros elementos de

comunicação que informam em qual ponto o código está. Dessa forma, é possível detectar qual foi o ponto que o software foi interrompido.

Nos casos quando se utilizam os registros de memória ou o processo interativo é necessário preparar a aplicação que será utilizada para fazer a depuração, para a preparação é preciso compilar esse software de maneira que ele tenha diversas informações de debug que normalmente não são colocadas junto ao binário para não influenciar no desempenho e no tamanho final do binário (GANSSLE, 2003). E ainda, quando se utiliza o "*dump*" é necessário gravar esses dados no sistema de armazenamento, o que pode causar um uso excessivo. No caso da instrumentação ocupa pouco espaço extra comparado aos outros modelos de depuração, todavia é necessário um mecanismo de comunicação em que seja possível enviar essas informações (sistema de arquivos compartilhados, rede, etc.) (SALLY, 2010).

O processo de depuração é feito de acordo com as possibilidades que o sistema embarcado possui. Utilizar registros de memória ou o processo interativo varia de acordo com a forma de construção. A opção de instrumentação é mais direta e pode ser considerada a simples obtenção de log.

Como exemplo, o código do Quadro 4.1 apresenta um código fonte feito em C que possui um defeito. Referente a este quadro, as linhas 1 e 2 representam a inclusão das bibliotecas do sistema operacional, a linha 4 declara o início da aplicação, a linha 6 representa o tamanho máximo do buffer, a linha 7 faz a alocação de memória do tamanho bufferSize e guarda referência de memória na variável buffer, e por fim entre a linha 10 e 13 é inserido o valor da multiplicação da variável i por ela mesma em cada posição alocada. O defeito está na comparação do comando *for* da linha 10, em que se acessa uma posição de memória maior que o valor alocado (bufferSize).

Quadro 4.1 | Código que possui um defeito no acesso à memória

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
	int bufferSize = 10;
	int *buffer = (int*) malloc(bufferSize*sizeof(int));
	int i = 0;

	for(i = 0; i <= bufferSize*10;i++)
	{
		buffer[i] = i*i;
	}
}
```

Fonte: elaborada pelo autor.

Para compilar e executar a aplicação do Quadro 4.1 se utiliza o comando gcc (GNU C compiler), considerando uma instalação de GNU/Linux Ubuntu ou debian se utiliza o comando:

- gcc exemploFalha.c -o programa

Em outros sistemas operacionais é possível executar processos semelhantes, no caso do Microsoft Windows é possível utilizar o Cygwin (Disponível em: <https://www.cygwin.com/>. Acesso em: 1 ago. 2018), com esse software são instalados terminais, compiladores e depurados idênticos ao do GNU/Linux, porém portados para o Windows. Em sistemas baseados em Mac OS se pode instalar os mesmos depuradores utilizando o projeto Homebrew (Disponível em: <https://brew.sh/>. Acesso em: 8 jan. 2018), novamente, fornecendo as mesmas ferramentas que são disponibilizadas para o debain ou Ubuntu.

No comando o parâmetro exemploFalha.c informa o código a ser compilado, no caso o exemploFalha.c é o arquivo que contém o código do Quadro 4.1, a opção -o informa qual será o nome do binário executável, que neste caso é o “executavel”. Ao executar esse código o sistema operacional informará a mensagem “Segmentation Fault” ou Falha de segmentação. Essa falha indica que se tentou acessar um segmento de memória que não era permitido ou não existia. Com isso, a aplicação é encerrada pelo sistema operacional. Neste processo, o sistema operacional ou o programa não informa de nenhuma maneira em qual linha ocorreu o erro.

Para iniciar a depuração interativo é necessário que no processo de compilação seja incluído os símbolos que informa as linhas do código fonte e outras informações que possibilitem que quando o software for executado seja possível descobrir onde ocorreu o defeito gerador da falha. Para isso se deve adicionar a opção -g na compilação (YAGHMOUR, 2003):

- gcc -g exemploFalha.c -o programa

Com isso os símbolos de depuração são adicionados, todavia na execução no caso de falha ainda não será informado onde o problema ocorreu. Todavia, agora é possível utilizar uma ferramenta para descobrir onde está o defeito. A ferramenta em questão é o gdb (GNU Debugger) (ZHANGJIN, 2013), neste caso para executar a aplicação é necessário informar que essa se apoiará no GDB, conforme a Figura 4.1 apresenta. Considerando a Figura 4.1, na primeira linha se compila o código fonte do Quadro 4.1 com a opção -g, na segunda linha é feita a execução do software com o suporte do GDB utilizando o comando:

- gdb/programa

Nas quatro últimas linhas informa o erro apresentando a falha de segmentação, e ainda informa que o problema ocorreu na linha 12. Mesmo depois da falha ocorrida, ainda é possível executar comandos para GDB que podem informar valores de variáveis ou origem das chamadas de função que ocasionaram o erro.

Figura 4.2 | Execução do software com o suporte do GDB

```
> gcc -g exemploFalha.c -o programa
> gdb ./programa
GNU gdb (Debian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./programa...done.
(gdb) r
Starting program: /home/fandri/programa

Program received signal SIGSEGV, Segmentation fault.
0x00005555555546ff in main () at exemploFalha.c:12
12                      buffer[i] = i*i;
(gdb) █
```

Fonte: elaborada pelo autor.

Um exemplo de outras formas de debug é utilizar a porta serial ou uma porta USB. No caso do GNU/Linux ao fazer a criação e conexão se utiliza o /dev/ttyUSB0, com isso todas as informações serão enviadas pela porta serial. Para isso é necessário fazer acesso da /dev/ttyUSB0 de forma idêntica ao um arquivo de texto, como exemplos na linguagem C: open("/dev/ttyUSB0," O_RDWR | O_NONBLOCK).

Ainda no GDB, após uma falha ocorrer é possível utilizar o comando bt, esse comando apresenta o backtrace da falha, informando quais foram as chamadas de função que resultaram na falha (YAGHMOUR, 2013). Na Figura 4.2 é possível verificar a execução do comando bt apresentando que a falha foi na função main do programa que foi executado.

Todo o processo de depuração é demorado e agrega mais complexidade ao projeto. O fato de considerar a depuração em um projeto já consiste na necessidade de instalar mais ferramentas no target. Qual é o *trade-off* (considerar as opções e verificar quais são os ganhos e perdas) de se considerar diversas opções de debug em um projeto?

Figura 4.3 | Execução do comando bt do GDB

```
(gdb) bt
#0  0x00005555555546ff in main () at exemploFalha.c:12
(gdb) 
```

Fonte: elaborada pelo autor.

Existem diversos outros comandos que o GDB possui, com eles é possível marcar pontos no código onde a execução será suspensa e com isso é possível analisar os valores das variáveis (YAGHMOUR, 2003):

- break ou b – suspende a execução do código em uma certa linha de código, sendo a informação nome do arquivo e a linha: b exemploFalha.c:12.

- clear – remove o breakpoint: clear exemploFalha.c:12.
- continue ou c: força o código voltar a executar depois de um breakpoint.
- next ou n: faz o depurador executar a próxima linha de execução.
- backtrace full: exibe o valor das variáveis locais.

O método de depuração local com o GDB é eficiente em arquiteturas de sistemas embarcados em que no *target* é possível instalar as ferramentas de depuração, compiladores e outros elementos. Todavia, em certos cenários em que a memória do sistema embarcado é limitada se faz necessário uma abordagem diferente.

A depuração remota consiste em utilizar o *gdbserver* instalado no sistema embarcado. Com ele é possível iniciar uma sessão de depuração em que toda a saída será enviada pela rede ou por uma porta serial. Na Figura 4.3 é possível ver a criação da sessão com o comando (YAGHMOUR, 2003):

- *gdbserver* 192.168.0.1:6789 programa

O comando *gdbserver* recebe como parâmetro o IP e porta que ficará disponível para fazer a conexão e o parâmetro programa define qual software será executado.

Figura 4.4 | Criação de sessão de depuração

```
> gdbserver 192.168.0.1:6789 programa
Process programa created; pid = 5955
Listening on port 6789
```

Fonte: elaborada pelo autor.

A Figura 4.4 apresenta como se deve fazer a conexão para a depuração, após o comando gdb programa se utiliza o comando:

- target remote 192.168.0.1:6789

Com esse comando feito dentro do GDB é possível fazer a depuração remota.

Figura 4.5 | Depuração remota

```
> gdb programa
GNU gdb (Debian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
```

```
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from programa...done.
(gdb) target remote 192.168.0.1:6789
Remote debugging using 192.168.0.1:6789
Reading /lib64/ld-linux-x86-64.so.2 from remote target...
warning: File transfers from remote targets can be slow. Use "set sysroot" to access files locally instead.
Reading /lib64/ld-linux-x86-64.so.2 from remote target...
Reading symbols from target:/lib64/ld-linux-x86-64.so.2...Reading symbols from /usr/lib/debug/.build-id/6f/150f33b150d6a81
e26a425dd47d713d00f2d29.debug...done.
done.
0x00007ffff7dd9c20 in _start () from target:/lib64/ld-linux-x86-64.so.2
(gdb) bt
```

Fonte: elaborada pelo autor.

O processo de depuração de um sistema embarcado é uma etapa sensível para que o produto final seja entregue sem grandes problemas ou ainda problemas que não podem ser resolvidos depois da produção.

Existem diversas outras maneiras de fazer depuração de um sistema embarcado, alguns projetos são de grande valia:

- <https://elinux.org/Electric_Fence>. Acesso em: 18 dez. 2017 – Detecção de vazamento de memória.
- <http://valgrind.org/>. Acesso em: 18 dez. 2017 – Detecção de vazamento de memória e análise de thread.
- strace – esse comando é padrão nos pacotes GNU/Linux, ele mostra todos os arquivos que um binário utiliza na execução.
- <https://cygwin.com/install.html>. Acesso em: 8 jan. 2018 - Como instalar o cygwin provendo as ferramentas de compilação em depuração para o Windows.
- <https://brew.sh/>. Acesso em: 8 jan.2018 – Informações para instalhar o GDB e GCC para o Mac OS.

Sem medo de errar

Neste cenário a empresa está produzindo um sistema autônomo de processamento de áudio para detectar eventos em volta da cidade. Gritos, disparo de armas de fogo ou outros eventos podem

ser detectados por esse sistema. Todavia, interrupções e travamentos do sistema foram relatados pela equipe de testes, tais eventos que tiveram problemas:

1. Ao deixar o sistema executando por cerca de 4 horas o software simplesmente para de funcionar e para restaurar é necessário retirar a fonte de energia elétrica e ligá-lo novamente.
2. O sistema está em funcionamento normal, porém, ao receber dois eventos sonoros com um intervalo menor que 1 minuto fica extremamente lento.

Nestes dois casos é necessário averiguar o que está ocorrendo com o sistema, considerando diversos processos de depuração é necessário encontrar a técnica mais indicada para encontrar a origem do problema. Para a situação 1 o sistema fica por um certo tempo em execução e depois seu processamento é interrompido, o processo interativo é uma opção válida. Porém, deixar o sistema em depuração causa mais lentidão fazendo com que os problemas demorem para aparecer, e ainda, quando a falha ocorrer o sistema não está disponível para interação dado a natureza desse problema. O mais indicado é utilizar o sistema de instrumentação, pois com ele é possível colocar diversas mensagens que são escritas em arquivo de texto (instrumentação), dessa forma quando o sistema travar e parar de responder é possível fazer a reinicialização do sistema e analisar em qual ponto o código para de responder através do log.

No caso 2 o sistema continua passível de comunicação e as falhas ocorrem sem grandes intervalos, a instrumentação é válida. Porém, o processo de depuração interativo é indicado, pois o sistema ainda responde a comandos e é possível simular a entrada de dados (gerar os áudios semelhantes aos sons reais) e detectar onde o código está gerando os problemas. Dessa forma, é possível detectar de forma mais rápida o problema.

Os dois cenários representam casos que ocorrem de forma rotineira em sistemas embarcados, com isso é necessário sempre ter em mente quais ferramentas são passíveis de utilização e qual deles utilizar.

Avançando na prática

Sistema de controle para esterilização via ultravioleta

Descrição da situação-problema

Um sistema de controle para uma câmara de esterilização via ultravioleta está apresentando um comportamento errático. O sistema funciona de forma correta durante dias, seu processo consiste em acionar a luz ultravioleta da câmara com potência de 25% por 30 minutos, potência em 50% por 1 hora e potência de 100% por 1 hora. Todavia, diversas vezes o sistema ficou parado na etapa de 50% de potência. Nesse cenário é necessário identificar qual é a melhor abordagem de depuração para detectar esse problema.

Resolução da situação-problema

No cenário que descreve um processo que fica "preso" em certos passos é interessante descobrir em qual porção de código ele ficou parado e tentar identificar qual foi a razão. Para isso, a melhor forma é a instrumentação do código fonte declarando certas etapas para escrever um arquivo de texto como no Quadro 4.2 que apresenta em suas linhas 1 a 3 as bibliotecas utilizadas pelo código, a linha 7 e 8 imprimem na tela memória (se pode redirecionar pelo sistema operacional para um arquivo de texto) que será alocado uma porção de memória nas linhas 12 e 13, indicando que será feito um processamento.

Quadro 4.2 | Código que possuir instrumentação para depuração

```
#include <stdio.h>
#include <stdlib.h>
#DEFINE debug 1
int main()
{
      int bufferSize = 10;
      if(debug)
        printf("Alocando memória\n");
      int *buffer = (int*) malloc(bufferSize*sizeof(int));
      int i = 0;
```

```
        if(debug)
          printf("Alocando memória\n");

        for(i = 0; i <= bufferSize;i++)
        {
              buffer[i] = i*i;
        }
  }
```

Fonte: elaborada pelo autor.

Esse processo consiste em escrever mensagens via log, rede, porta serial ou outras formas para informar onde o código parou. Como exemplo, se no arquivo de log estiver apenas escrito "Alocando memória" se sabe que o processo ficou parado nessa etapa. Com a informação onde o sistema não pode prosseguir é possível analisar a porção que está gerando essa interrupção.

Faça valer a pena

1. O processo de depuração em sistema embarcado é diferente de um computador de uso geral, pois as limitações de hardware e interfaces são maiores em produtos embarcados. Dessa forma, algumas técnicas são úteis para superar essas limitações de analisar os problemas que estão ocorrendo no firmware.

Dos processos de debug, assinale quais desses comandos do GNU/Linux podem ser úteis no processo.

a) gdb e gdbserver
b) mkfs e utf8
c) grub e lilo
d) format e dir
e) free e top

2. Existem diversas formas de depurar um código fonte e cada técnica é indicada para determinados cenários. Os processos de depuração interativa são de grande valia, pois conseguem mostrar as variáveis, linha de código e outras informações da memória do software em execução. No

processo interativo é necessário inserir no código elementos que enviem informação para o disco ou rede para informar os dados do código.

Em um cenário em que o sistema não responde a nenhum comando ou interação que são enviados:
I – A depuração interativa é indica por ser mais precisa.
II – A instrumentação do código fonte é o ideal, pois será possível ter as informações mesmo após uma reinicialização forçada.
III – A depuração por dump de memória é interessante, porém o travamento do sistema pode influenciar na informação que será gravada.
IV – A depuração nesse cenário não é indicada, pois falhas em hardware não são detectáveis por software.
V – O GDB é um sistema de depuração por instrumentação.
Assinale quais informações são verdadeiras:

a) Apenas a afirmação I e II são verdadeiras.
b) Apenas a afirmação III e V são verdadeiras.
c) Apenas a afirmação I é verdadeira.
d) Apenas a afirmação II e III são verdadeiras.
e) Apenas a afirmação I, II, III e IV são verdadeiras.

3. No processo de depuração interativa existem diversos comandos que podem fazer a leitura de variáveis de um código fonte, esses comandos são úteis para ler os conteúdos variáveis, inspecionar linha a linha e outras opções.

Dos comandos abaixo pertencentes do GDB tem a função de mostrar as linhas de código e funções de um código fonte que apresentaram o erro em sistema.

a) bt
b) st
c) n
d) b
e) q

Seção 4.2

Dispositivos embarcados em tempo real

Diálogo aberto

Na seção anterior o objetivo foi detectar a origem dos problemas do sistema embarcado, para isso foi necessário utilizar ferramentas de depuração que geravam formas de encontrar os defeitos no código. Nesta etapa do estudo o foco serão os sistemas operacionais de tempo real para sistemas embarcados. Os sistemas de tempo real são dedicados para que o tempo de resposta seja regular, dessa forma é possível controlar os equipamentos que necessitam precisão tal como sistemas de segurança para maquinário industrial, sistemas de controle de equipamentos, sistemas médicos e outros. Os sistemas de tempo real podem ser comparados a metrônomos (equipamentos de música que emitem um som com uma taxa por minuto, por exemplo 100 batidas por minutos), eles são capazes de manter um ritmo preciso para que todas as pessoas que estão tocando a música fiquem sincronizadas. Essa parte dos estudos trará os conhecimentos necessários para a aplicação de sistemas embarcados em ambientes mais complexos em que a capacidade técnica é mais valorizada.

Uma empresa que desenvolve sistemas de segurança para prensas hidráulicas na produção de artefatos de borracha (equipamento que gera grande pressão e calor para produzir peças de borracha como vedações, pneus e outros) está com um novo projeto para um sistema monitorado por sensores de barreira por infravermelho em que se o operador invadir certas áreas da prensa durante a operação o sistema deve imobilizar a máquina, retirando toda a pressão e calor. Um dos membros da equipe de projetos afirma que não é necessário um sistema operacional especial para que o sistema embarcado seja produzido nesse sistema autônomo de monitoramento. A sua tarefa é convencer por meio de uma apresentação de 5 minutos com slides que mostrem os aspectos de um sistema operacional de tempo real para essa aplicação, contendo os itens:

1. Comparação entre um sistema operacional geral e de tempo real.
2. Problemas que podem causar ao ser utilizado um sistema operacional geral.
3. Custos para a implementação de um sistema de tempo real.

Não pode faltar

As aplicações para um sistema embarcado são diversificadas, podemos encontrar sistemas embarcados em carros, aviões, equipamentos militares, relógios e muitos outros equipamentos. Todavia, podemos dividir os sistemas pelos seus requisitos dentre os quais podem ser considerados requisitos de tempo de resposta. O tempo de resposta em sistemas embarcados é um requisito que deve ser considerado no início do projeto. O tempo de resposta é uma das características dos sistemas de tempo real, segundo Ganssle, (2003) os sistemas de tempo real são representados por qualquer sistema (embarcado ou não) onde o tempo é um requisito, segundo Yaghmour (2003), os sistemas em tempo real são caracterizados por executar suas tarefas de forma logicamente correta e além disso seu tempo de resposta fazer parte da avaliação se o sistema está operando de forma correta.

De acordo com Ganssle (2003), existe uma pergunta simples que define se um sistema tem de ser de tempo real ou não: uma pergunta atrasada é ruim ou até pior que uma resposta errada? Outras formas de avaliar é considerar o tempo de resposta relacionado à aplicação do projeto: se existe atraso em uma resposta, alguém poderá morrer ou se ferir ou durante a execução pode interromper toda a produção? Em caso de respostas positivas, o sistema é considerado um sistema hard real-time, ou seja, onde qualquer atraso na resposta é considerado problema sério na operação do sistema embarcado.

Outra forma de sistema embarcado de real-time são sistemas que possuem uma interface com usuários (que dependem de algum tipo de resposta), todavia, o atraso não é considerado um problema que comprometa a operação do sistema embarcado. Nesses casos, o atraso no tempo de resposta é definido pelos requisitos

do sistema, no qual as especificações podem variar segundo sua aplicação (GANSSLE, 2008).

O tempo de resposta de um sistema está relacionado ao processamento, todavia, nos sistemas de tempo real a resposta de um sistema é desenhada pelas prioridades que o sistema define à aplicação. A aplicação pode demorar o tanto de tempo que seus requisitos definam, porém, o sistema operacional tem de entregar prontamente (sem esperar ou processos ou chamadas do sistema operacional) os dados ou sinais que a aplicação precisa para garantir a execução de forma correta dado os requisitos (NOERGAARD, 2012).

Exemplificando

O projeto de um sistema embarcado que controla a abertura de um portão de condomínio residencial pela leitura de impressão digital recebeu um requisito de tempo de resposta. O tempo das seguintes operações somado não pode ultrapassar o tempo de 5 segundos:

- Leitura da impressão digital.
- Processamento da impressão digital.
- Acionamento do atuador para abertura da porta.

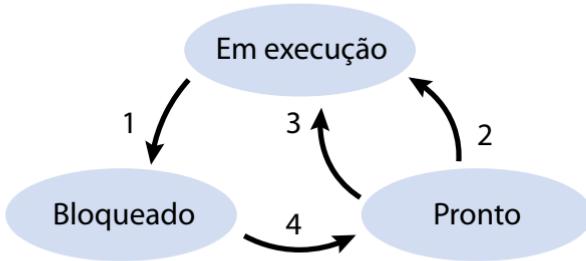
Todavia, no final do projeto se descobriu que devido à resolução da imagem fornecida pelo leitor de impressão digital o processamento tomaria cerca de 4 segundos e com isso o tempo de resposta do sistema seria de 6 segundos. Nesse cenário o sistema para abertura de portas é caracterizado como soft real-time, pois o atraso no processamento não é causador de grandes problemas, o atraso de poucos segundos não torna o sistema inviável.

Dessa forma, podemos classificar os sistemas embarcados de tempo real em duas características: soft ou hard. O Quadro 4.3 apresenta uma comparação entre as duas classificações (SALLY, 2010).

Quadro 4.3 | Comparação entre sistema de tempo real *Hard* e *Soft*

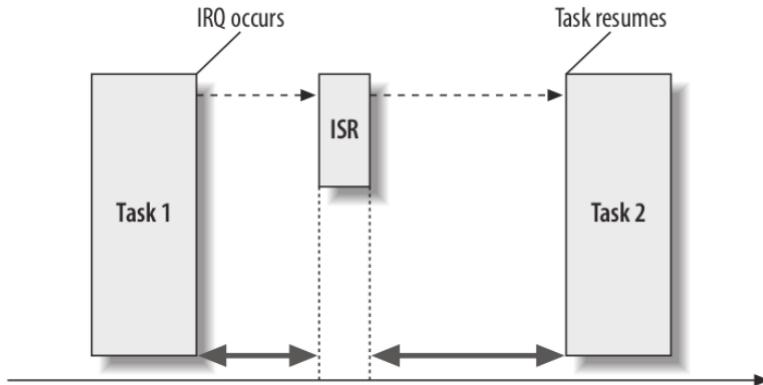
	Hard Real Time	Soft Real Time
Pior tempo de desempenho do sistema	O pior tempo de desempenho é o mesmo	O tempo de desempenho varia em torno da média

	do tempo médio de desempenho.	dado os requisitos.
Tempo de entrega dos resultados ultrapassado	Se o tempo de entrega não for atendido o resultado do trabalho é descartado.	A entrega de resultados atrasados pode degradar a qualidade do serviço oferecido.
Consequenciais	Resultados atrasados podem causar danos irrecuperáveis, tais como a morte, ferimentos ou danos mecânicos.	Degradação da qualidade do serviço oferecido que podem ser recuperados ou tolerados.
Aplicação típica	Aviões, equipamentos médicos, sistema de transporte ou sistema de proteção em máquinas industriais.	Sistema de redes, telecomunicações ou entretenimento.

Fonte: adaptado de Sally (2010, p. 258).

A classificação de um sistema de tempo real está relacionada à aplicação que foi desenvolvida e também com o sistema embarcado que demora para entregar os dados para a aplicação. Dados seus requisitos, os sistemas de tempo real soft podem ter atrasos na resposta sem causar problemas e os hard real-time, caso ocorram atrasos as consequências serão devastadoras. Os sistemas de tempo real são relacionados a garantias do tempo de entrega e não à velocidade de processamento, ou seja, o tempo de entrega de sinais e execução das aplicações é feita de maneira organizada e com tempo de respostas constantes.

O tempo de resposta de um sistema operacional está relacionado em como as tarefas são enviadas pelo escalonador para serem executadas pelo processador. Mesmo em um sistema com diversos processadores, os processos (programas que estão sendo executados pelo sistema operacional) são enviados pelo escalonador (parte do sistema operacional que faz a troca de tarefas a serem executadas pelos processadores) para que suas instruções sejam executadas pelo processador. Todavia, quando um processo necessita a execução de algum tipo de interface de entrada e saída, escrita ou leitura de memória ou outros eventos externos é necessário fazer tais requisições e suspender momentaneamente a execução do software. A Figura 4.6 apresenta a sequência de estados de um processo, o

processo tem seu início na etapa "Pronto", que significa que tem todos os requisitos para executar, com isso ele vai para a etapa "Em execução" e suas instruções são executadas pelo processador. Do estado "Em execução" se pode ir para o estado de pronto novamente quando o escalonador decide que esse processo já executou o tempo determinado ou ir para o estado de "Bloqueado" quando é necessário algum tipo de operação externa ao processo como leitura de arquivos, dados da rede e outros. Quando o processo bloqueado recebe os dados que precisa, volta para o estado de "Pronto" que pode voltar ao estado "Em execução" (TANENBAUM, 2010).

Figura 4.6 | Sequência de estados de um processo

1. O processo bloqueia aguardando uma entrada
2. O escalonador seleciona outro processo
3. O escalonador seleciona esse processo
4. A entrada torna-se disponível

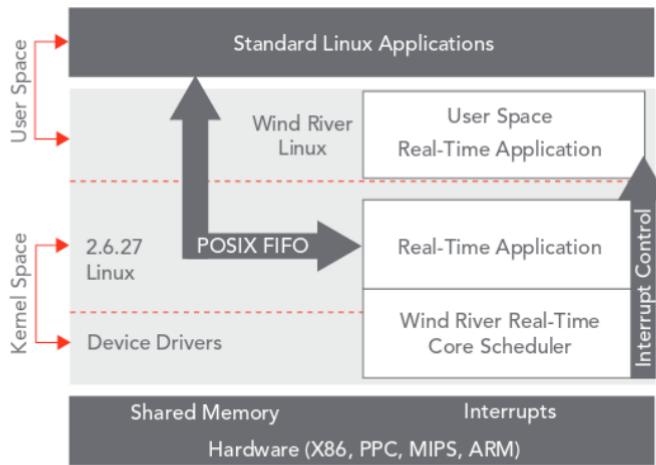
Fonte: Tanembaum (2010, p. 54).

Os sistemas em tempo real possuem a mesma sistemática sobre o escalonador. Todavia, existem diversas considerações nesse processo para que o sistema seja tratado como um sistema operacional de tempo real. Todas essas operações de troca de estado do escalonador consomem tempo de processamento, e ainda, existem momentos que certos processos não podem ser interrompidos como escritas não finalizadas em disco ou em memória. Portanto, em um sistema operacional é necessário tratar diversos eventos para que seja possível a execução de diversos processos ao mesmo tempo. A Figura 4.7 apresenta dois tipos de atrasos que podem ocorrer na execução de alguns softwares, a Task 1 necessitou de algum dado externo gerando um *Interrupt request* (IRQ), quando isso ocorre o escalonador deve

parar uma tarefa e quando o *interrupt service routine* (ISR) informa que algum dado ou sinal está pronto para a execução de uma tarefa a Task2 volta a executar (YAGHMOUR, 2003).

Figura 4.7 | Tipos de atrasos ou delays de um sistema operacional

Fonte: Yaghmour (2013, p. 356).

Em um sistema de tempo real é necessário que esses atrasos sejam muito bem alinhados para que o tempo de resposta das aplicações seja constante. Para que isso seja possível é necessário que o sistema operacional esteja pronto e tenha os recursos necessários com o objetivo de garantir que quando uma aplicação necessite entrar em execução possa fazer prontamente. Para essa tarefa é necessário que o sistema operacional tenha (SALLY, 2015):

1. Controle de prioridades de processo que seja respeitado por todas as aplicações.
2. Escalador preemptivo (que pode interromper tarefas) preciso onde as tarefas de alta prioridade sejam executadas primeiro que as de baixa prioridade.
3. Sistema de controle de entrada e saída com restrições de tempo.

O GNU/Linux clássico não possui as características necessárias para atender os requisitos de um sistema operacional hard real-time, dessa forma existem alguns projetos que fazem adendos ao Linux para que seja possível a execução, tais projetos:

1. Xenomai Real-Time System - <https://xenomai.org/start-here/> .Acesso em: 28 dez. 2017.
2. Real-Time Linux - <https://rt.wiki.kernel.org/index.php/Main_Page>. Acesso em: 28 /de. 2017.
3. WindRiver VxWorks - <https://www.windriver.com/products/product-overviews/2691-VxWorks-Product-Overview/>. Acesso em: 28 dez. 2017.
4. Yocto Project - <https://www.yoctoproject.org/>. Acesso em: 29 dez. 2017.

Todas essas soluções conseguem garantir tempos regulares de interrupções, sistemas preemptivos de processos e ainda fornecem uma *Application programming interface* (API) para que as chamadas de tempo sejam tradadas de forma correta e com as prioridades necessárias de um sistema de tempo real. A Figura 4.8 apresenta a camada de aplicações do sistema operacional (*Standard Linux Applications*) que se conecta a pilha clássica na camada de *User Space* que por sua vez se conecta a parte de real time e o escalonador e por fim no hardware. Para o tratamento de eventos de tempo real existem ligações diretas para que a aplicação consiga acesso mais rápido ao hardware.

Figura 4.8 | Parte central do sistema de tempo real do *Wind river*

Fonte: *Wind River* (2009, p. 2).

Em um projeto de sistema embarcado para o controle da parte de segurança de máquinas de grande porte para corte de papel se deve utilizar um sistema operacional hard real-time. Quais seriam os efeitos da utilização de um sistema soft real-time nesse projeto? Qual seria o pior cenário dado essa escolha errada de tipo de sistema operacional?

Como exemplo de utilização de um sistema de tempo real utilizaremos o Real-Time Linux ou também chamado com RT Linux. O Linux não vem com as implementações do RT Linux, com isso é necessário aplicar um conjunto de patch (conjunto de código que fazem correções ou adendos em um software). Para isso é necessário buscar o código fonte do kernel e do RT Linux e aplicar os adendos. Em um ambiente GNU/Linux, a Figura 4.9 apresenta o processo de criação de um diretório com o comando mkdir, a utilização desse diretório com o comando cd; com o diretório se utiliza o comando wget para recuperar os *patchs* que serão aplicados no kernel Linux. A Figura 4.10 apresenta o processo de download do kernel Linux utilizando o comando wget.

Figura 4.9 | Download das modificações do kernel Linux para sistema de tempo real

```
> mkdir rt
> cd rt
> wget https://www.kernel.org/pub/linux/kernel/projects/rt/2.6.29/patch-2.6.29.6-rt24.bz2
--2017-12-28 15:58:52--  https://www.kernel.org/pub/linux/kernel/projects/rt/2.6.29/patch-2.6.29.6-rt24.bz2
Resolving www.kernel.org (www.kernel.org)... 147.75.196.57, 2604:1380:1:3600::3
Connecting to www.kernel.org (www.kernel.org)|147.75.196.57|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 973320 (951K) [application/x-bzip2]
Saving to: 'patch-2.6.29.6-rt24.bz2'

patch-2.6.29.6-rt24.bz2       100%[================================================>] 950.51K   880KB/s    in 1.1s

2017-12-28 15:58:54 (880 KB/s) - 'patch-2.6.29.6-rt24.bz2' saved [973320/973320]
```

Fonte: elaborada pelo autor.

Figura 4.10 | Download do kernel Linux

```
> wget http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.29.5.tar.bz2
--2017-12-28 15:59:11--  http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.29.5.tar.bz2
Resolving kernel.org (kernel.org)... 198.145.29.83
Connecting to kernel.org (kernel.org)|198.145.29.83|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://kernel.org/pub/linux/kernel/v2.6/linux-2.6.29.5.tar.bz2 [following]
--2017-12-28 15:59:11--  https://kernel.org/pub/linux/kernel/v2.6/linux-2.6.29.5.tar.bz2
Connecting to kernel.org (kernel.org)|198.145.29.83|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.29.5.tar.bz2 [following]
--2017-12-28 15:59:12--  https://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.29.5.tar.bz2
Resolving www.kernel.org (www.kernel.org)... 147.75.196.57, 2604:1380:1:3600::3
Connecting to www.kernel.org (www.kernel.org)|147.75.196.57|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 56550101 (54M) [application/x-bzip2]
Saving to: 'linux-2.6.29.5.tar.bz2'

linux-2.6.29.5.tar.bz2        100%[================================================>]  53.93M   954KB/s    in 59s

2017-12-28 16:00:11 (943 KB/s) - 'linux-2.6.29.5.tar.bz2' saved [56550101/56550101]
```

Fonte: elaborada pelo autor.

Após a recuperação dos códigos fonte do *kernel* Linux e do RT Linux é necessário, conforme a Figura 4.11, descompactar o código do *kernel* do Linux com o comando tar e com o comando *bunzip2* descompactar os patchs do RT Linux e com o comando *patch* aplicar as modificações no Linux.

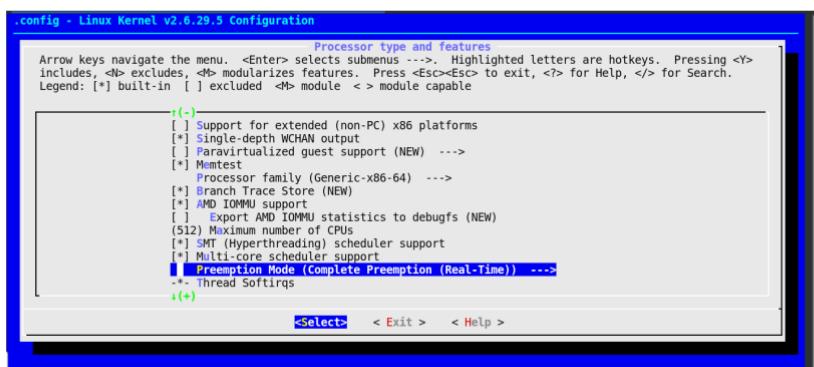



Figura 4.11 | Descompactação e aplicação dos *patchs* do RT Linux

```
> tar -xf linux-2.6.29.5.tar.bz2
> cd linux-2.6.29.5/
> bunzip2 -c ../patch-2.6.29.6-rt24.bz2 | patch -p 1
patching file Documentation/ABI/testing/debugfs-kmemtrace
patching file Documentation/DMA-API.txt
patching file Documentation/DocBook/genericirq.tmpl
patching file Documentation/cputopology.txt
patching file Documentation/feature-removal-schedule.txt
patching file Documentation/ftrace.txt
patching file Documentation/kernel-parameters.txt
patching file Documentation/kmemcheck.txt
```

Fonte: elaborada pelo autor.

Com a aplicação dos *patchs* é possível configurar o Linux para que utilize o escalonador de tempo real, e para que seja possível acessar as opções de configuração do Linux é necessário utilizar o comando *make menuconfig*, com isso é possível acessar as opções de configuração como na Figura 4.12. Entre diversas opções, se pode selecionar a *Preemption Mode*.

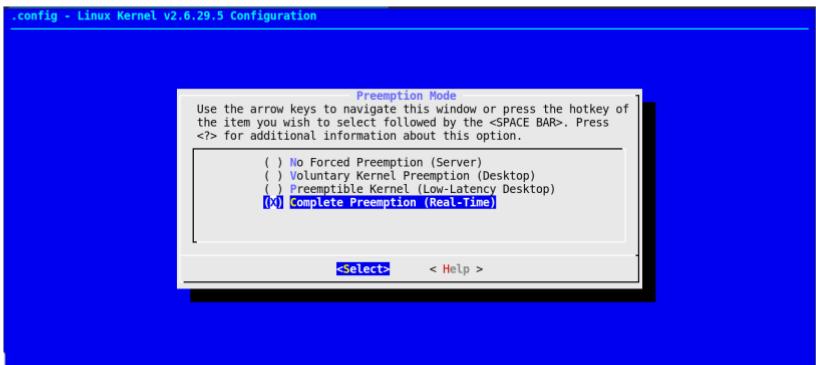



Figura 4.12 | Configuração para compilação do kernel Linux

```
.config - Linux Kernel v2.6.29.5 Configuration
                         Processor type and features
 Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.  Pressing <Y>
 includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
 Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                    ↑(-)
                    [ ] Support for extended (non-PC) x86 platforms
                    [*] Single-depth WCHAN output
                    [ ] Paravirtualized guest support (NEW)  --->
                    [*] Memtest
                        Processor family (Generic-x86-64)  --->
                    [*] Branch Trace Store (NEW)
                    [*] AMD IOMMU support
                    [ ]   Export AMD IOMMU statistics to debugfs (NEW)
                    (512) Maximum number of CPUs
                    [*] SMT (Hyperthreading) scheduler support
                    [*] Multi-core scheduler support
                        Preemption Mode (Complete Preemption (Real-Time))  --->
                    -*- Thread Softirqs
                    ↓(+)

                         <Select>    < Exit >    < Help >
```

Fonte: elaborada pelo autor.

Conforme a Figura 4.13, na opção *Preemption Mode*, é possível utilizar opção *Complete Preemption* onde o *kernel* do Linux com os patchs do RT Linux poderá utilizar um escalonador que pode garantir os intervalos dos processos, garantindo o tempo de resposta de um processo.

Figura 4.13 | Seleção da opção de Real-Time do Linux

Fonte: elaborada pelo autor.

Com a configuração do *kernel* feita é possível executar o comando make que fará a compilação, com a imagem do *kenel* com as opções é possível integrar essa imagem como visto em etapas anteriores. Todavia, apenas ter um kernel feito para tempo real é o necessário para que a aplicação seja realmente de tempo real. A utilização da API de processos para tempo real é mandatória para garantir que os recursos do *kernel* sejam aproveitados de forma correta. Nas diversas aplicações, se utiliza threads que representa linhas de execução independentes a linha principal do processo.

No Quadro 4.4 é possível analisar um exemplo da utilização da API de threads, para um sistema de tempo real baseado no RT Linux. Nas linhas 1 até 6 representam as bibliotecas que serão utilizadas no código, entre as linhas 10 e 17 está o código que será executado pela thread que faz a leitura de um sensor temperatura, na linha 26 o comando *pthread_attr_init* define que a variável attr será utilizada para receber os parâmetros da thread, na linha 28 se define com o comando *pthread_attr_setschedpolicy* que o escalonador utilizará a política de primeiro que entra (a ser processador) é o primeiro que sai (que é processado) ou First In First Out considerando a prioridade de cada thread, na linhas 30 define a prioridade em 80 (onde 0 é a menor e 99 é a maior), na linha 32 se fixa essa opção na thread, a linha 34 força que as opções escolhidas não sejam sobrescritas por parâmetros padrão, na linha 36 se cria a thread que faz com que a função thread_funcao inicie e faça as leitura dos sensores, e por fim na linha 38 se espera que a thread acabe seu processamento.

A simples definição da prioridade do thread e as configurações do *kernel* garantem que a execução do thread seja feita de forma regular.

Quadro 4.4 | Exemplos de código que utiliza a API de threads para tempo real do kernel RT Linux

```
#include <limits.h>
#include <pthread.h>
#include <sched.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>

int processa = 1;

void *thread_funcao (void *data)
{
    while(processa == 1)
    {
        int valor = leituraSensorTemperatura();
    }
    return NULL;
}

int main(int argc, char* argv[])
{
    struct sched_param param;
    pthread_attr_t attr;
    pthread_t thread;
    int ret;

    ret = pthread_attr_init(&attr);

    ret = pthread_attr_setschedpolicy (&attr,
SCHED_FIFO);

    param.sched_priority = 80;

    ret = pthread_attr_setschedparam(&attr, &param);

    ret = pthread_attr_setinheritsched(&attr, PTHREAD_-
EXPLICIT_SCHED);

    ret = pthread_create(&thread, &attr, thread_funcao,
NULL);

    ret = pthread_join(thread, NULL);

    return ret;
}
```

Fonte: adaptado de <https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/application_base>. Acesso em: 29 dez. 2017.

Com todas essas configurações e modificações nos códigos é possível garantir que a execução de um código seja feita de forma regular e com tempo de entrega previsível. Dessa forma, sendo possível controlar equipamentos em que o tempo de resposta é essencial.

Pesquise mais

Existem diversas fontes de informação sobre sistema embarcado de tempo real, alguns sites que são de grande valia:

1. Xenomai Real-Time System - <https://xenomai.org/start-here/>. Acesso em: 28 dez. 2017.
2. Real-Time Linux - <https://rt.wiki.kernel.org/index.php/Main_Page>. Acesso em: 28 dez. 2017;
3. Windriver VxWorks - <https://www.windriver.com/products/product-overviews/2691-VxWorks-Product-Overview/>. Acesso em: 28 dez. 2017.
4. <http://www.ee.nmt.edu/~rison/ee352_spr12/Getting_Started_with_RTLinux.pdf>. Acesso em: 29 /dez. 2017.
5. <https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/application_base>. Acesso em: 29 dez. 2017.
6. Yocto Project - <https://www.yoctoproject.org/>. Acesso em: 29 dez. 2017.
7. Build root - <https://buildroot.org/>. Acesso em: 29 dez. 2017.
8. <https://developer.microsoft.com/en-us/windows/iot>. Acesso em: 17 jan. 2018.
9. <https://www.embarcados.com.br/real-time-no-windows-embedded-compact/> Acesso em: 17 jan./ 2018.
10. <https://www.insight.tech/healthcare/leverage-windows-10-real-time-extensions-for-medical-iot>. Acesso em: 17 jan. 2018.
11. <https://www.youtube.com/watch?v=E06ecYle2mo>. Acesso em: 17 jan. 2018.

Sem medo de errar

O cenário consiste na aplicação de um sistema embarcado para a construção de um sistema autônomo para monitoramento de uma prensa hidráulica. Esse sistema deve monitorar com barreiras de infravermelho se o operador invadir certa área da máquina que pode gerar perigo, em que a máquina será imobilizada e toda a pressão e calor removidos. Nesse projeto é necessário convencer a empresa a utilizar um sistema operacional de tempo real em uma apresentação contendo:

1. Comparação entre um sistema operacional geral e de tempo real.
2. Problemas que podem causar ao ser utilizado um sistema operacional geral.
3. Custos para a implementação de um sistema de tempo real.

Na comparação entre sistemas de tempo real e um de uso geral a principal diferença se dá pelos seguintes itens:

- Sistemas operacionais de uso geral possuem intervalos irregulares de execução de suas tarefas.
- A *Application programming interface* de um sistema operacional de uso geral não possui as funções para ações em tempo real.

Em relação aos problemas que podem ocorrer ao utilizar um sistema de uso geral no cenário apresentado:

- Sem intervalos regulares da execução das aplicações e atrasos não controlados na execução dos programas, podendo causar atrasos na leitura dos sensores ou no acionamento de atuadores o que podem ser fatais.
- O atraso na leitura dos sensores pode inviabilizar uma imobilização da prensa no tempo correto, causando ferimento ou morte para o operador.

Sobre os custos de utilização de um sistema de tempo real:

- Existem opções *open-source* para sistema de tempo real como o RT Linux.
- Caso seja necessário suporte e uma curva de aprendizado menor se pode utilizar versões pagas (Windriver VxWorks) ainda projetos open-source que são fomentados por empresas como o *Yocto Project*.

Avançando na prática

Sistema de detecção de irregularidades no asfalto para veículos de passeio

Descrição da situação-problema

O sistema para detecção de irregularidades no asfalto para veículos de passeio utiliza uma câmera de alta resolução instalada na frente do carro que analisa o asfalto ou outros obstáculos na frente do veículo. Com essa câmera é possível detectar problemas que estão a cerca de 2km, com isso é possível avisar o condutor para tomar alguma providência como diminuir a velocidade ou até parar o carro. O projeto utiliza um sistema de tempo real para que seja possível minimizar os atrasos entre o processamento das imagens e o acionamento da interface entre o usuário. Todavia, mesmo utilizando o sistema de tempo real estão ocorrendo atrasos quando os obstáculos estão próximo ao veículo.

Resolução da situação-problema

Nesse cenário é preciso verificar se o código do software que está sendo desenvolvido para os sistemas embarcados utilizou a *application programming interface* (API) para tempo real provida pelo sistema operacional. Com a utilização dessa API se torna possível informar o sistema operacional qual é a prioridade dos processos e com isso executar as aplicações em intervalos regulares. Outro problema está relacionado à capacidade de processamento do hardware para analisar as imagens em alta resolução, um sistema

operacional em de tempo real não implica que o processo será mais rápido e sim ocorrerá em intervalos determinísticos.

Faça valer a pena

1. Os sistemas de tempo real possuem diversas aplicações, umas que recebem destaque são sistemas de segurança para máquinas industriais. Máquinas industriais de diversos portes possuem sistemas embarcados autônomos que fazem o monitoramento de equipamentos e possuem o papel essencial de evitar que o operador ou pessoas próximas sejam feridas pelo maquinário.

Dada as afirmações abaixo:
I – Sistemas de tempo real são desenhador para executar de forma mais rápida.
II – Os delays de um sistema de tempo real são aleatórios.
III - O escalonador de um sistema embarcado tem papel essencial na execução de tarefas de forma determinística.
IV – O Linux necessita de adendos para que seja considerado de tempo real.

a) Apenas a afirmação I é correta.
b) Apenas a afirmação I e II são corretas.
c) A afirmação I, II e III são corretas.
d) Todas as afirmações estão corretas.
e) Apenas as afirmações III e IV são corretas.

2. Em um sistema de tempo real é necessário que escalonador tenha capacidade de definir quais processos devem ser executados no momento correto, dessa forma é possível garantir que o tempo de resposta das aplicações seja regular. Todavia, ainda é necessário que as aplicações sejam preparadas de forma relacionada ao sistema operacional de tempo real.

Dos elementos abaixo, qual deles está relacionado às adaptações que o software deve receber para possa utilizar os recursos de tempo real do sistema operacional.

a) API para Thread e controle de prioridade
b) Mutex
c) IP
d) Sistema de interface
e) Thread

3. Em um sistema de tempo real, os delays são provocados por diversas fontes onde o escalonador de processos se organiza para que esses atrasos sejam regulados e determinísticos. Os intervalos sendo determinísticos garantem que seja possível garantir quando a aplicação executará.

Quais dos itens abaixo são responsáveis pela geração de atrasos em um sistema operacional?

a) *Jitter* de rede
b) Falha na memória
c) Controle de processos
d) *Interrupt request* e troca de contexto
e) PID

Seção 4.3

Sensores e atuadores embarcados

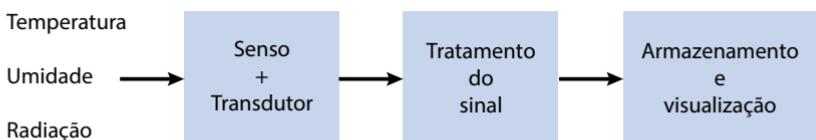
Diálogo aberto

No estudo da seção passada foram analisados os sistemas embarcados de tempo real, os quais têm aplicações específicas parar cenários de segurança de máquinas, área hospitalar e outros. A decisão de utilizar um sistema de tempo real é feita quando o atraso em alguma resposta pode causar danos ao patrimônio ou ferimentos às pessoas. Agora o foco são os sensores e atuadores que farão a leitura de temperatura, humidade e outros e também os atuadores como motores que farão alguma ação controlada pelo sistema embarcado. Como comparação, podemos citar que os sensores são os sentidos dos seres humanos como audição, visão e outros, o sistema embarcado é o cérebro que comanda as ações, e os membros (braços e pernas) são os atuadores que fazem as ações controladas pelo cérebro.

Uma empresa de transporte público está recebendo diversas reclamações dos passageiros sobre a forma que estão sendo dirigidos seus ônibus. Muitos dos passageiros apontam que os motoristas aceleram, freiam e fazem curvas com os seus veículos de forma abrupta causando desconforto e viagens mais cansativas. Todavia, devido ao custo elevado não é possível instalar câmeras de vídeo em todos os veículos e ainda por questões financeiras é necessário que os custos sejam reduzidos. Para esse projeto se deve pesquisar sensores que consigam detectar variações de movimento e atuadores para informar que a forma de dirigir está causando desconforto. Todo esse processo deve ter como base o custo de um sistema com câmera de vídeo próximo de R$ 500,00 por unidade, os requisitos são:

- Custo por equipamento para detecção e análise da forma de dirigir do motorista, devendo ser reduzido, pois será instalado em todos os veículos.
- O sistema deve conseguir detectar algum tipo de direção abrupta e avisar a central via sistema sem fio desse tipo de evento.

- Deve conter um sistema que acione um sinal sonoro ou luminoso avisando o motorista sobre o estilo de direção que está sendo utilizado.

Esse projeto chegou para a sua empresa e é necessário realizar uma apresentação com um slide contendo uma descrição do sistema em forma de tópicos listando:

- Descrição do funcionamento.
- Materiais utilizados.
- Custos estimados dos componentes.

Não pode faltar

Os sistemas embarcados possuem diversas aplicações, é possível utilizar tais sistemas em relógios, sistemas de segurança, equipamento médico e muitos outros exemplos (YAGHMOUR, 2003). Todavia, mesmo tais equipamentos tendo um uso tão diversificado possuem um elemento em comum, utilizam interfaces para sensores, atuadores (elementos que fazem algum tipo de ação, por exemplo motor de corrente alternada), transdutores (componentes que transformam uma grandeza física em sinal elétrico normalizado) e outros elementos (CRUZ, 2008).

Muitos dos sistemas embarcados possuem diversos tipos de interfaces para controlar ou ler esses elementos de acesso ou medida do mundo real. A Figura 4.14 apresenta como é o processo de leitura de grandeza física, que pode ser a temperatura do ar, água, etc., umidade, radiação ou outros valores. Para essa leitura são necessários um sensor e um transdutor que farão a medida da grandeza física, esse valor é enviado para tratamento que pode ser ampliado ou normalizado, e por fim é enviado para armazenamento e visualização (BISHOP, 2016).

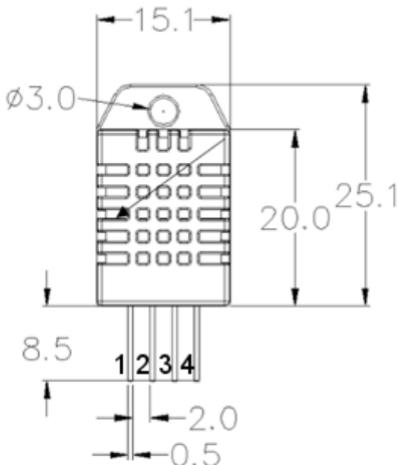
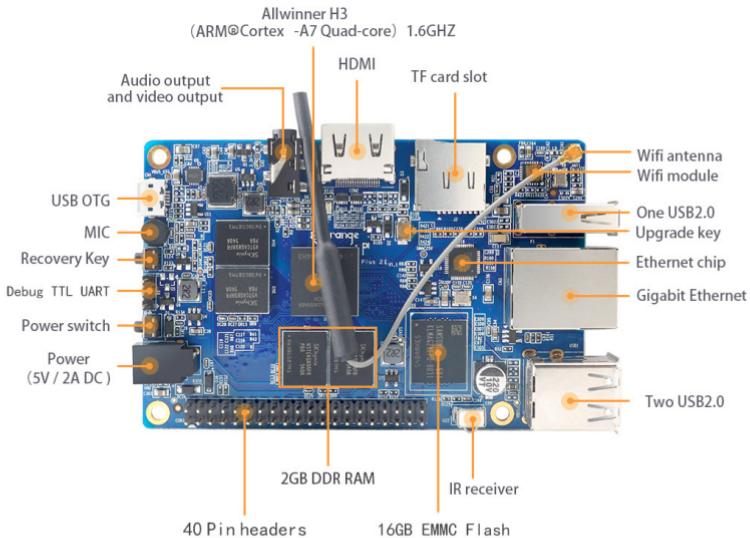



Figura 4.14 | Processo de leitura de uma grandeza física

Fonte: elaborada pelo autor.

Nesse processo, o sistema embarcado pode estar presente no tratamento do sinal ou esse processo pode ser feito por algum componente eletrônico; a parte de armazenamento e visualização é feita pelo sistema embarcado. Um exemplo de um sensor para a leitura da temperatura e umidade é o circuito integrado DHT22 (AOSONG ELECTRONICS, 2017). Segundo a Figura 4.15 o pino número 1 é alimentação, o pino 2 é responsável por enviar os dados, o pino 3 não é utilizado e o pino 4 é a ligação do neutro.

Figura 4.15 | Esquema do sensor DHT22

Fonte: adaptado de Aosong Electronics (2018, p. 4).

Para fazer a leitura desses dados é necessário conectar esse sensor em alguma placa que possui uma entrada digital (sinal que tem variação discreta que representa os dados). Como exemplos podemos utilizar Orange Pi Plus 2E que é uma placa que possui diversas interfaces e suporte para sistema operacional. A Figura 4.16 apresenta a Orange Pi Plus 2E com todos os elementos de interface e características da placa, os itens que a placa possui em destaque são:

- 2GB DDR3 RAM.
- CPU H3 Quad-core Cortex-A7 H.265/HEVC 4K.
- Realtek RTL8189ETV, IEEE 802.11 b/g/n.
- Interface de propósito geral de entrada de saída ou General-purpose input/output (GPIO).

- Armazenamento interno de 16GB e entrada para cartão de memória.
- Rede Ethernet 10/100/1000M/s.

Figura 4.16 | Visão superior da placa Orange Pi Plus 2E

Fonte: <http://www.orangepi.org/orangepiplus2e/>. Acesso em: 12 abr. 2018.

No caso da ligação do sensor é necessário utilizar GPIO para fazer leitura dos dados, na Figura 4.17 é possível verificar qual é a função de cada pino da GPIO da placa. Cada pino da placa possui uma função específica (BROADCOM, 2012):

- 3.3 e 5V são fontes de energia.
- GND são os pinos com o neutro.
- SDA e SCL são para comunicação I2C.
- **GPCLK saída ou entrada de sinais de clock.**
- **UART, TXD e RXD são para as portas serias como RS232.**
- **IO-0 até IO-6 são entradas e saídas digitais.**
- SPI, MOSI/MISO/CS0/CS1 são para comunicação via serial bus.
- SCLK é a saída de clock para serial bus.

A GPIO é a parte de um sistema embarcado que possui diversos tipos de entradas e saídas que fazem a comunicação entre outros dispositivos, sensores, atuadores e outros elementos. Dessa forma, em um projeto é necessário verificar quantos dispositivos serão controlados e quantas portas a placa do sistema embarcado possui.

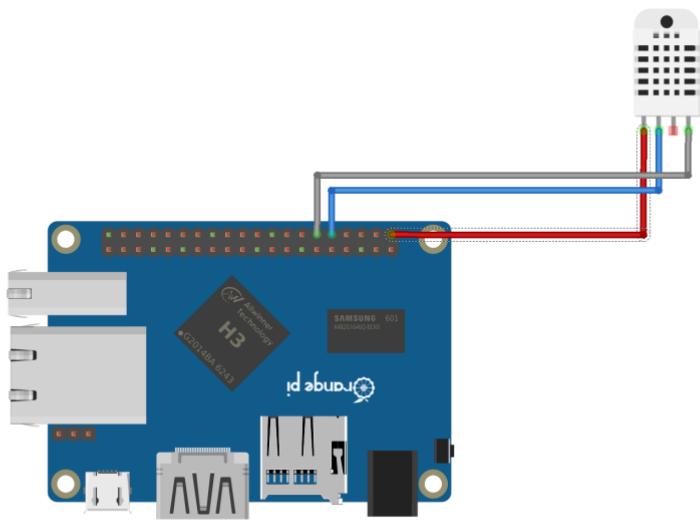



Figura 4.17 | Função dos pinos da GPIO do Orange Pi Plus 2E

	P01	P02	
3.3V			5V
SDA			5V
SCL			GND
GPCLK			TXD
GND			RXD
10-0			10-1
10-2			GND
10-3			10-4
3.3V			10-5
MOS1			GND
M1SO			10-6
SCLK			CS0
GND			CS1
	P25	P26	

Fonte: adaptado de <http://www.orangepi.org/Docs/Pindefinition.html>. Acesso em: 12 abr. 2018.

Para fazer a comunicação entre o Orange Pi e o DHT22 é necessário, conforme Figura 4.18, ligar o pino 1 do DHT22 no canal 1 da GPIO, o pino 2 do DHT22 (dados) no canal 7 (GPCLK) que fará o papel de receber os dados do sensor/transdutor, e por fim o quarto pino do DHT22 no canal 6 da GPIO.

Figura 4.18 | Esquema de ligação entre o sensor DHT22 e o Orange pi

Fonte: elaborado pelo autor.

Após o processo de ligação elétrica é necessário fazer o software que fará a leitura do sensor e tratará a informação. Para isso é necessário configurar a biblioteca de GPIO do Orange Pi. Com o sistema operacional já instalado (como visto em seções anteriores, mais informações em <http://www.orangepi.org/quickstart/start_ddcaed797a20bade87c2a52c91a206.html>. Acesso em: 3 /jan. 2018) é necessário fazer o download no Orange pi das bibliotecas através do sistema de controle de versão pelo comando: git clone <https://github.com/duxingkei33/orangepi_PC_gpio_pyH3> e com comando git clone <https://github.com/ionutpi/DHT22-Python-library-Orange-PI>. A Figura 4.19 apresenta o resultado do processo.

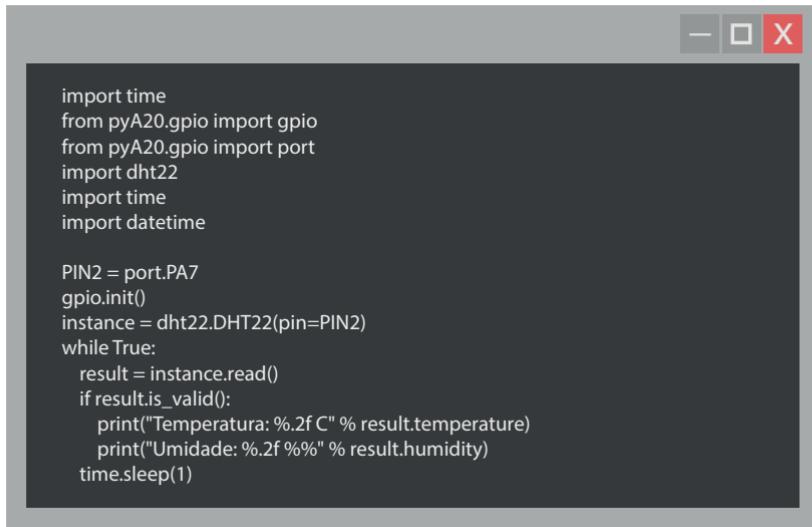



Figura 4.19 | Recuperação do código da biblioteca de controle da GPIO do Orange Pi

```
> git clone https://github.com/duxingkei33/orangepi_PC_gpio_pyH3
Cloning into 'orangepi_PC_gpio_pyH3'...
remote: Counting objects: 61, done.
remote: Total 61 (delta 0), reused 0 (delta 0), pack-reused 61
Unpacking objects: 100% (61/61), done.
> cd orangepi_PC_gpio_pyH3
> sudo python setup.py install
> git clone https://github.com/ionutpi/DHT22-Python-library-Orange-PI
```

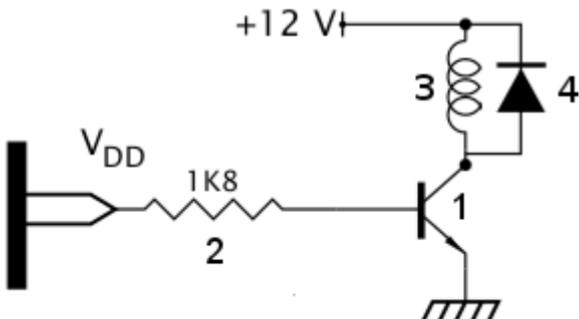
Fonte: elaborada pelo autor.

Com as bibliotecas configuradas é possível fazer um código que o Orange Pi executará para fazer a leitura do sensor. O Quadro 4.5 apresenta um exemplo de como fazer a leitura do sensor, a linha 1 até a linha 6 fazem a declaração das bibliotecas que serão utilizadas nos códigos, as linhas 8 a 10 fazem a configuração inicial e define qual porta da GPIO será utilizada para fazer a leitura e entre as linhas 11 a 16 é feita a leitura dos valores e impressão na tela.

Quadro 4.5 | Exemplo de código para fazer a leitura de um sensor

```
import time
from pyA20.gpio import gpio
from pyA20.gpio import port
import dht22
import time
import datetime

PIN2 = port.PA7
gpio.init()
instance = dht22.DHT22(pin=PIN2)
while True:
   result = instance.read()
   if result.is_valid():
      print("Temperatura: %.2f C" % result.temperature)
      print("Umidade: %.2f %%" % result.humidity)
   time.sleep(1)
```

Fonte: adaptado de <https://github.com/ionutpi/DHT22-Python-library-Orange-PI>. Acesso em: 12 abr. 2018.

Sobre atuadores também é possível utilizar a GPIO para fazer seu acionamento, como atuadores podemos considerar motores, mas para outros exemplos também são interessantes como válvulas solenoides (válvula que abre fecha por controle de um campo magnético), atuadores pneumáticos (controlados por ar) ou

hidráulicos (controlado por óleo) (HELFRICK, 1994). A grande maioria desses atuadores utiliza corrente e tensão muito superior ao que uma interface digital que GPIO dos sistemas embarcado disponibiliza. Para que seja possível utilizar esses equipamentos controlados por uma placa como o Orange Pi é necessário fazer um circuito chamado drive. Conforme a Figura 4.20, esse circuito utiliza um transistor, marcado como 1 na figura, elemento da eletrônica que funciona como um interruptor e aciona eletricamente (RAZAVI,2010) esse transistor, que é ligado na porta digital da GPIO. Por sua vez, esse transistor não permite que a energia flua da fonte de alimentação (+12V) para a bobina da relé (marcado como 3 na Figura 4.20) até que a porta da GPIO seja acionada. O elemento 2 da Figura 4.20 é um resistor para limitar a corrente da GPIO evitando problema de aquecimento ou queima, e o item 4 é um diodo que não permite o retorno de tensão como forma de proteção do circuito.

A GPIO de um sistema embarcado em diversos casos não é preparada para o controle de equipamento de potência, é necessário utilizar outros elementos de controle para que seja possível acionar motores de grande porte, válvulas e outros elementos. Dessa forma, o projeto de controle de equipamentos de potência deve fazer parte do projeto do sistema embarcado?

Figura 4.20 | Circuito para acionamento de equipamentos de potência

Fonte: adaptada de Katzen (2005, p. 292).

Para fazer o acionamento desse relé é necessário acionar a porta digital da GPIO. No Quadro 4.6 o código entre as linhas 1 a 3 são definidas às bibliotecas que serão utilizadas, as linhas 5 e 6 fazem a configuração da GPIO e define o pino 7 como saída, a linha 8 aciona a saída da GPIO que por sua vez aciona a relé, a linha 10 faz com que o sistema espere 2 segundos e a linha 12 desliga o pino da GPIO que desliga a relé.

Quadro 4.6 | Controle do pino da GPIO

```
from pyA20.gpio import gpio
from pyA20.gpio import port
from time import sleep

gpio.init()
gpio.setcfg(port.PA7, gpio.OUTPUT)

gpio.output(port.PA7, gpio.HIGH)

sleep(2)

gpio.output(port.PA7, gpio.LOW)
```

Fonte: adaptado de <http://www.instructables.com/id/Orange-Pi-One-Python-GPIO-basic/>. Acesso em: 12 abr. 2018.

Com esse tipo de acionamento é possível controlar diversos outros elementos, como as válvulas solenoides exemplificados na Figura 4.21, quando acionada permite a passagem de líquidos ou ar. Para que seja possível o acionamento é necessário alimentar eletricamente a válvula e interromper um dos fios no relé conforme a Figura 4.20.

Exemplificando

Existem diversos outros tipos de atuadores e sensores, tais como:

1. Sensores:
    - Contato
    - Proximidade
    - Força

- Peso
- Temperatura
- Umidade
- Radiação

2. Atuadores:

- Lineares hidráulicos, pneumáticos ou eletromagnéticos
- Motores de corrente contínua e alternada
- Servo motores
- Motores de passo

Figura 4.21 | Válvula solenoide

Fonte: <http://www.jefferson.ind.br/conteudo/valvula-solenoide.html>. Acesso em: 12 abr. 2018.

O exemplo da válvula solenoide é apenas um dos elementos que podem ser acionados por um sistema embarcado. As possibilidades de controle de um equipamento são muito grandes, considerando contatores (equipamentos eletromecânicos para acionamento de equipamento de potência, relés de estados sólidos (componente para acionamento de potência baseado em semicondutor), acelerômetros (equipamentos que podem informar a direção e intensidade de movimento) e outros elementos (MOHAN, 2014).

Seguem alguns links que são úteis para estudar sobre o Orange Pi:

- <http://www.orangepi.org/quickstart/start_ddcaed797a20bade87c2a52c91a206.html>. Acesso em: 3 jan. 2017.
- <https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf>. Acesso em: 03 jan. 2017.
- <http://www.jefferson.ind.br/conteudo/valvula-solenoide.html>. Acesso em: 12 abr. 2018.
- <http://www.linak.com.br/products/linear-actuators.aspx>. Acesso em: 12 abr. 2018.

Sem medo de errar

A sua empresa foi contrata para desenvolver um sistema para detectar se um motorista de ônibus está dirigindo de forma abruta. Esse sistema tem por objetivo informar uma central e ao motorista como os passageiros estão se sentindo reportando que a viagem está sendo conturbada. Os custos desse equipamento devem ser baixos para que possa ser instalado em todos os ônibus da empresa.

Para apresentar esse projeto é necessária uma apresentação de slides contendo a descrição do sistema, os materiais utilizados e o custo estimado. Para isso a apresentação deve conter em relação ao material utilizado:

- Sistema embarcado com sistema operacional com suporte à rede wifi e GPIO (Orange Pi).
- Acelerômetro para 3 eixos com interface serial.
- Relés, transistor, resistores e diodos para interface entre o sinal sonoro e luminoso.
- Lâmpada vermelha e sirene para sinal ao motorista.

Sobre o funcionamento:

- O sistema embarcado com sistema operacional utilizará a GPIO para ler os dados do acelerômetro.

- Com os dados do acelerômetro é possível detectar se os movimentos estão sendo abruptos.
- Com a detecção do estilo de direção estressante o sistema utilizará a GPIO para acionar os sinais e o wifi para avisar a central.

Custos (preços levantados em loja on-line como aliexpress.com, acesso em 12 abr. 2018):

- Orange Pi: R$ 120,00.
- Acelerômetro: R$ 20,00.
- Relés, transistor, resistores e diodos: R$ 20,00.
- Lâmpada e sirene: R$ 25,00.

Avançando na prática

Título da nova situação-problema

Sistema de contadores de utilização para manutenção preventiva

Descrição da situação-problema

Um parque de diversões está preocupado com a manutenção de seus brinquedos. Alguns deles possuem catracas ou algum tipo de controle de acesso que é capaz de contar quantas pessoas já utilizaram o equipamento. Todavia, alguns equipamentos não possuem catracas ou qualquer tipo de controle de acesso e um novo regulamento na presidência da empresa regulamenta que se deve saber quantas pessoas já utilizaram o equipamento para planejar manutenções e ainda ter informações para gerar campanhas de marketing.

A sua empresa foi contrata para propor uma solução para o projeto com um sistema embarcado. Esse sistema deve ser capaz de contar quantas pessoas passaram de certa área do brinquedo que caracteriza uma utilização. Você deve descrever um sensor e um sistema embarcado que possa fazer essa contagem e gerar relatórios.

Resolução da situação-problema

Para fazer a contagem das pessoas é necessário algum tipo sensor que consiga perceber que uma pessoa cruzou certa área. Um sensor indicado para esse cenário é uma barreira ótica conforme Figura 4.21. Esse sensor possui dois feixes infravermelho que quando interrompidos é possível detectar que uma pessoa ultrapassou certa área e ainda em qual direção ele se dirigiu (entrou ou saiu do brinquedo). Para utilizar esse sensor se deve ligá-lo em uma entrada digital do GPIO de uma placa para sistema embarcado como o Orange Pi.

Figura 4.22 | Barreira ótica com capacidade de definir qual direção foi o movimento

Fonte: <http://www.decibel.com.br/index.php?route=product/product&path=2_56&product_id=109>. Acesso em: 12 abr. 2018.

Faça valer a pena

1. Os sensores e atuadores para sistemas embarcados possuem as mais diversas aplicações: em sistemas para controle de irrigação de jardins, sistemas médicos até projetos militares. As diferentes aplicações possuem suas peculiaridades em características sobre a qualidade, precisão ou preço.

Qual é a parte do sistema embarcado que é utilizada para fazer a leitura ou utilização de um sensor ou atuador?

a) GPIO
b) TCP
c) Resistores
d) UDP
e) Power set

2. Para que seja possível fazer a leitura de um sensor são necessários dois elementos para que os sensores dessa grandeza física sejam possíveis de se transformar em algum tipo de informação formatada. Esses dois elementos têm a capacidade de deixar os sinais prontos para a utilização de um usuário.

Na utilização de um sistema embarcado, quais são os dois elementos necessários para que seja possível utilizar um sensor através da GPIO?

a) Processamento e envio
b) Transdutor e normalização
c) Transdutor e processamento
d) Normalização e armazenamento
e) Processamento e armazenamento

3. Diversas placas de sistema embarcado possuem uma GPIO para garantir acesso a sensores e atuadores. Todavia, diversos atuadores utilizam muito mais corrente elétrica que a GPIO pode suportar, sendo necessário utilizar circuitos junto à GPIO para fazer a utilização de atuadores.

Quais componentes eletrônicos têm papel principal para que seja possível que uma GPIO possa acionar um motor de corrente alternada de potência elétrica elevada?

a) Transistor e relé
b) Contator e capacitor
c) Diodo e transistor
d) Resistor e relé
e) Capacitor e transistor

Referências

AOSONG ELECTRONICS. **Digital-output relative humidity & temperature sensor/ module DHT22 (DHT22 also named as AM2302)**. [S.l.: s.n.], 2017.

BISHOP, Owen. **Electronics - A First Course**. 2 ed. [S.l.]: Elsevier, 2006. 235 p.

BROADCOM, BCM2835 ARM Peripherals. [S.l.: s.n.], broadcom, 2012. 205 p.

CRUZ, Eduardo Alves, JUNIOR, Salomão Choueri. **Eletrônica Aplicada.** 2. ed. Érica, 2008.

GANSSLE, Jack G.; BARR, Michael. **Embedded systems dictionary**. [S.l.]: Taylor, 2003.

GANSSLE, Jack. **The art of designing embedded systems**. [S.l.]: Elsevier Science, 2008.

HELFRICK, Albert D.; COOPER, Wiliam D. **Instrumentação eletrônica moderna e técnicas de medição**. [S.l.]: Pretince-Hall Brasil, 1994. 336 p.

MOHAN, Ned. **Eletrônica de Potência - Curso Introdutório**. LTC, 2014.

NOERGAARD, Tammy. **Embedded systems architecture:** a comprehensive guide for engineers and programmers. [S.l.]: Newnes, 2012. 672 p.

KATZEN, Sid. **The Quintessential PIC Microcontroller**. [S.l.]: Springer-Verlag, 2000. 506 p.

RAZAVI, Behzad. **Fundamentos de Microeletrônica**. LTC, 2010.

OLIVEIRA, André de; ANDRADE, Fernando de. **Sistemas Embarcados - Hardware e Firmware na Prática**. 2. edo. Érica, 2010.

SALLY, Gene. **Pro Linux Embedded Systems**. [S.l.]: Apress, 2010. 550 p.

TANENBAUM, Andrew S. **Sistemas Operacionais Modernos**. [S.l.]: Pearson, 2010. 672 p.

YAGHMOUR, Karim. **Building embedded linux systems.** [S.l.]: O'Reilly Media, Inc., 2003. 416 p.YAGHMOUR, Karim. Building embedded linux systems. [S.l.]: O'Reilly Media, Inc., 2003. 416 p.

ZHANGJIN, Wu; CAO, Ziqiang. **Instant Optimizing Embedded Systems using Busybox**. [S.l.]: ProQuest, 2013. 59 p.